On the Complexity of Real Solving Bivariate Systems*

[Extended abstract]

Dimitrios I. Diochnos
National University of Athens
Athens, Hellas
d.diochnos(at)di.uoa.gr

Ioannis Z. Emiris
National University of Athens
Athens, Hellas
emiris(at)di.uoa.gr

Elias P. Tsigaridas
LORIA-INRIA Lorraine
Nancy, France
elias.tsigaridas(at)loria.fr

ABSTRACT

We consider exact real solving of well-constrained, bivariate systems of relatively prime polynomials. The main problem is to compute all common real roots in isolating interval representation, and to determine their intersection multiplicities. We present three algorithms and analyze their asymptotic bit complexity, obtaining a bound of $\widetilde{\mathcal{O}}_B(N^{14})$ for the purely projection-based method, and $\widetilde{\mathcal{O}}_B(N^{12})$ for two subresultants-based methods: these ignore polylogarithmic factors, and N bounds the degree and the bitsize of the polynomials. The previous record bound was $\widetilde{\mathcal{O}}_B(N^{14})$.

Our main tool is signed subresultant sequences, extended to several variables by binary segmentation. We exploit advances on the complexity of univariate root isolation, and extend them to multipoint sign evaluation, sign evaluation of bivariate polynomials over two algebraic numbers, and real root counting over an extension field. Our algorithms apply to the problem of simultaneous inequalities; they also compute the topology of real plane algebraic curves in $\widetilde{\mathcal{O}}_B(N^{12})$, whereas the previous bound was $\widetilde{\mathcal{O}}_B(N^{14})$.

All algorithms have been implemented in MAPLE, in conjunction with numeric filtering. We compare them against FGB/RS and SYNAPS; we also consider MAPLE libraries INSULATE and TOP, which compute curve topology. Our software is among the most robust, and its runtimes are within a small constant factor, with respect to the C/C++ libraries.

Categories and Subject Descriptors: F.2.1 [Analysis of Algorithms and Problem Complexity]; G.4 [Mathematical software]: Algorithm design and analysis;

General Terms: Algorithms, Experimentation, Theory

Keywords: polynomial system, real algebraic number, real solving, topology of real algebraic curve, maple

*All authors acknowledge partial financial support by FET-Open European Project ACS (Algorithms for complex shapes). The third author started work on this project while at University of Athens and INRIA Sophia-Antipolis. The third author is also partially supported by ARC ARCADIA.

ISSAC'07, July 29–August 1, 2007, Waterloo, Ontario, Canada.

1. INTRODUCTION

The problem of well-constrained polynomial system solving is fundamental. However, most of the algorithms treat the general case or consider solutions over an algebraically closed field. We focus on real solving of bivariate polynomials, in order to provide precise complexity bounds and study different algorithms in practice. We expect to obtain faster algorithms than in the general case. This is important in several applications ranging from nonlinear computational geometry to real quantifier elimination. We suppose relatively prime polynomials for simplicity, but this hypothesis is not restrictive. A question of independent interest is to compute the topology of a plane real algebraic curve.

Our algorithms isolate all common real roots inside non-overlapping rational rectangles, and output them as pairs of algebraic numbers; they also determine the intersection multiplicity per root. In this paper, $\mathcal{O}_B$ means bit complexity and $\widetilde{\mathcal{O}}_B$ means that we are ignoring polylogarithmic factors. We derive a bound of $\widetilde{\mathcal{O}}_B(N^{12})$, whereas the previous record bound was $\widetilde{\mathcal{O}}_B(N^{14})$ [12], see also [3], derived from the closely related problem of computing the topology of real plane algebraic curves, where N bounds the degree and the bitsize of the input polynomials. This approach depends on Thom's encoding. We choose the isolating interval representation, since it is more intuitive, it is used in applications, and demonstrate that it supports as efficient algorithms as other representations. In [12] it is stated that "isolating intervals provide worst [sic] bounds". Moreover, it is widely believed that isolating intervals do not produce good theoretical results. Our work suggests that isolating intervals should be re-evaluated.

Our main tool is signed subresultant sequences (closely related to Sturm-Habicht sequences), extended to several variables by the technique of binary segmentation. We exploit the recent advances on univariate root isolation, which reduced complexity by 1-3 orders of magnitude to $\widetilde{\mathcal{O}}_B(N^6)$ [8, 9, 10]. This brought complexity closer to $\widetilde{\mathcal{O}}_B(N^4)$, which is achieved by numerical methods [25].

In [17], 2×2 systems are solved and the multiplicities computed under the assumption that a generic shear has been obtained, based on [31]. In [34], 2×2 systems of bounded degree were studied, obtained as projections of the arrangement of 3D quadrics. This algorithm is a precursor of ours, see also [11], except that matching and multiplicity computation was simpler. In [22], a subdivision algorithm is proposed, exploiting the properties of the Bernstein basis, with unknown bit complexity, and arithmetic complexity based on the characteristics of the graphs of the polynomials. For

other approaches based on multivariate Sturm sequences the reader may refer to e.g. [21, 26].

Determining the topology of a real algebraic plane curve is a closely related problem. The best bound is $\widetilde{\mathcal{O}}_B(N^{14})$ [3, 12]. In [35] three projections are used; this is implemented in INSULATE, with which we make several comparisons. Work in [15] offers an efficient implementation of resultant-based methods. For an alternative using Gröbner bases see [6]. To the best of our knowledge the only result in topology determination using isolating intervals is [2], where a $\widetilde{\mathcal{O}}_B(N^{30})$ bound is proved.

We establish a bound of $\widetilde{\mathcal{O}}_B(N^{12})$ using the isolating interval representation. It seems that the complexity in [12] could be improved to $\widetilde{\mathcal{O}}_B(N^{10})$ using fast multiplication algorithms, fast algorithms for computations of signed subresultant sequences and improved bounds for the bitsize of the integers appearing in computations. To put our bounds into perspective, note that the input is in $\mathcal{O}_B(N^3)$, and the bitsize of all output isolation points for univariate solving is $\widetilde{\mathcal{O}}_B(N^2)$, and this is tight.

The main contributions of this paper are the following: Using the aggregate separation bound, we improve the complexity for computing the sign of a polynomial evaluated over all real roots of another (lem. 2.7). We establish a complexity bound for bivariate sign evaluation (th. 2.14), which helps us derive bounds for root counting in an extension field (th. 4.1) and for the problem of simultaneous inequalities (cor. 4.2). We study the complexity of bivariate polynomial real solving, using three projection-based algorithms: a straightforward grid method (th. 3.1), a specialized RUR approach (th. 3.5), and an improvement of the latter using fast GCD (th. 3.6). Our best bound is $\widetilde{\mathcal{O}}_B(N^{12})$; within this bound, we also compute the root multiplicities. Computing the topology of a real plane algebraic curve is in $\widetilde{\mathcal{O}}_B(N^{12})$ (th. 4.3).

We implemented in MAPLE a package for computations with real algebraic numbers and for implementing our algorithms. It is easy to use and integrates seminumerical filtering to speed up computation when the roots are well-separated. It guarantees exactness and completeness of results; moreover, the runtimes seem very encouraging. We illustrate it by experiments against well-established C/C++ libraries FGB/RS and SYNAPS. We also examine MAPLE libraries INSULATE and TOP, which compute curve topology. Our software is robust and effective; its runtime is within a small constant factor w.r.t. the fastest C/C++ library.

The next section presents basic results concerning real solving and operations on univariate polynomials. We extend the discussion to several variables, and focus on bivariate polynomials. The algorithms for bivariate solving and their analyses appear in sec. 3, followed by applications to real-root counting, simultaneous inequalities and the topology of curves. Our implementation and experiments appear in sec. 5. Ancillary results and omitted proofs are found in [7].

2. PRELIMINARIES

For $f \in \mathbb{Z}[y_1, \ldots, y_k, x]$, $\mathsf{dg}(f)$ denotes its total degree, while $\deg_x(f)$ denotes its degree w.r.t. x. $\mathcal{L}(f)$ bounds the bitsize of the coefficients of f (including a bit for the sign). We assume $\lg(\mathsf{dg}(f)) = \mathcal{O}(\mathcal{L}(f))$. For $\mathsf{a} \in \mathbb{Q}$, $\mathcal{L}(\mathsf{a})$ is the maximum bitsize of numerator and denominator. Let $\mathsf{M}(\tau)$ denote the bit complexity of multiplying two integers of size τ, and $\mathsf{M}(d, \tau)$ the complexity of multiplying two univariate polynomials of degrees $\leq d$ and coefficient bitsize $\leq \tau$. Using FFT, $\mathsf{M}(\tau) = \widetilde{\mathcal{O}}_B(\tau)$, $\mathsf{M}(d, \tau) = \widetilde{\mathcal{O}}_B(d\tau)$.

Let $f, g \in \mathbb{Z}[x]$, $\mathsf{dg}(f) = p \geq q = \mathsf{dg}(g)$ and $\mathcal{L}(f), \mathcal{L}(g) \leq \tau$. We use $\mathsf{rem}(f, g)$ and $\mathsf{quo}(f, g)$ for the Euclidean remainder and quotient, respectively. The *signed polynomial remainder sequence* of f, g is $R_0 = f$, $R_1 = g$, $R_2 = -\mathsf{rem}(f, g)$, $\ldots$, $R_k = -\mathsf{rem}(R_{k-2}, R_{k-1})$, where $\mathsf{rem}(R_{k-1}, R_k) = 0$. The *quotient sequence* contains $Q_i = \mathsf{quo}(R_i, R_{i+1})$, $i = 0 \ldots k-1$, and the *quotient boot* is $(Q_0, \ldots, Q_{k-1}, R_k)$.

Here, we consider signed subresultant sequences, which contain polynomials similar to the polynomials in the signed polynomial remainder sequence; see [33] for a unified approach to subresultants. They achieve better bounds on the coefficient bitsize and have good specialization properties. In our implementation we use Sturm-Habicht sequences, see e.g. [13]. By $\mathbf{SR}(f, g)$ we denote the signed subresultant sequence, by $\mathbf{sr}(f, g)$ the sequence of the principal subresultant coefficients, by $\mathbf{SQ}(f, g)$ the corresponding quotient boot, and by $\mathbf{SR}(f, g; \mathsf{a})$ the evaluated sequence over $\mathsf{a} \in \mathbb{Q}$. If the polynomials are multivariate, then these sequences are considered w.r.t. x, except if explicitly stated otherwise.

PROPOSITION 2.1. *[18, 19, 27] Assuming $p \geq q$, $\mathbf{SR}(f, g)$ is computed in $\widetilde{\mathcal{O}}_B(p^2 q\tau)$ and $\mathcal{L}(\mathbf{SR}_j(f, g)) = \mathcal{O}(p\tau)$. For any f, g, their quotient boot, any polynomial in $\mathbf{SR}(f, g)$, their resultant, and their* gcd *are computed in $\widetilde{\mathcal{O}}_B(pq\tau)$.*

PROPOSITION 2.2. *[18, 27] Let $p \geq q$. We can compute $\mathbf{SR}(f, g; \mathsf{a})$, where $\mathsf{a} \in \mathbb{Q} \cup \{\pm\infty\}$ and $\mathcal{L}(\mathsf{a}) = \sigma$, in $\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma + p^2\sigma)$. If $f(\mathsf{a})$ is known, then the bound becomes $\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma)$.*

When $q > p$, $\mathbf{SR}(f, g)$ is $f, g, -f, -(g \bmod (-f)) \ldots$, thus $\mathbf{SR}(f, g; \mathsf{a})$ starts with a sign variation irrespective of $\mathrm{sign}(g(\mathsf{a}))$. If only the sign variations are needed, there is no need to evaluate g, so prop. 2.2 yields $\widetilde{\mathcal{O}}_B(pq\tau + p^2\sigma)$. Let L denote a list of real numbers. $\mathtt{VAR}(L)$ denotes the number of (possibly modified, see e.g. [3, 13]) sign variations.

COROLLARY 2.3. *For any f, g, $\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{a}))$ is computed in $\widetilde{\mathcal{O}}_B(pq\tau + \min\{p, q\}^2\sigma)$, provided $\mathrm{sign}(f(\mathsf{a}))$ is known.*

We choose to represent a real algebraic number $\alpha \in \mathbb{R}_{alg}$ by the *isolating interval* representation. It includes a square-free polynomial which vanishes on α and a (rational) interval containing α and no other root.

PROPOSITION 2.4. *[8, 9, 10] Let $f \in \mathbb{Z}[x]$ have degree p and bitsize τ_f. We compute the isolating interval representation of its real roots and their multiplicities in $\widetilde{\mathcal{O}}_B(p^6 + p^4\tau_f^2)$. The endpoints of the isolating intervals have bitsize $\mathcal{O}(p^2 + p\,\tau_f)$ and $\mathcal{L}(f_{red}) = \mathcal{O}(p + \tau_f)$.*

The sign of the square-free part f_{red} over the interval's endpoints is known; moreover, $f_{red}(\mathsf{a}) f_{red}(\mathsf{b}) < 0$.

COROLLARY 2.5. *[3, 10] Given a real algebraic number $\alpha \cong (f, [\mathsf{a}, \mathsf{b}])$, where $\mathcal{L}(\mathsf{a}) = \mathcal{L}(\mathsf{b}) = \mathcal{O}(p\tau_f)$, and $g \in \mathbb{Z}[x]$, such that $\mathsf{dg}(g) = q, \mathcal{L}(g) = \tau_g$, we compute $\mathrm{sign}(g(\alpha))$ in bit complexity $\widetilde{\mathcal{O}}_B(pq \max\{\tau_f, \tau_g\} + p \min\{p, q\}^2 \tau_f)$.*

Prop. 2.4 expresses the state-of-the-art in univariate root isolation. It relies on fast computation of polynomial sequences and the Davenport-Mahler bound, e.g. [36]. The following lemma, derived from Davenport-Mahler's bound, is crucial.

LEMMA 2.6 (AGGREGATE SEPARATION). *Given $f \in \mathbb{Z}[x]$, the sum of the bitsize of* all *isolating points of the real roots of f is $\mathcal{O}(p^2 + p\,\tau_f)$.*

We present a new complexity bound on evaluating the sign of a polynomial $g(x)$ over a set of algebraic numbers, which have the same defining polynomial, namely over all real roots of $f(x)$. It suffices to evaluate $\mathbf{SR}(f,g)$ over all the isolating endpoints of f. The obvious technique, e.g. [10], is to apply cor. 2.5 r times, where r is the number of real roots of f. But we can do better by applying lem. 2.6:

LEMMA 2.7. *Let $\tau = \max\{p, \tau_f, \tau_g\}$. Assume that we have isolated the r real roots of f and we know the signs of f over the isolating endpoints. Then, we can compute the sign of g over all r roots of f in $\widetilde{\mathcal{O}}_B(p^2 q\tau)$.*

We discuss multivariate polynomials, using binary segmentation [27]. An alternative approach could be [16]. Let $f, g \in (\mathbb{Z}[y_1, \ldots, y_k])[x]$ with $\mathsf{dg}_x(f) = p \geq q = \mathsf{dg}_x(g)$, $\mathsf{dg}_{y_i}(f) \leq d_i$ and $\mathsf{dg}_{y_i}(g) \leq d_i$. Let $d = \prod_{i=1}^k d_i$ and $\mathcal{L}(f), \mathcal{L}(g) \leq \tau$. The y_i-degree of every polynomial in $\mathbf{SR}(f,g)$, is bounded by $\mathsf{dg}_{y_i}(\mathtt{res}(f,g)) \leq (p+q)d_i$. Thus, the homomorphism $\psi : \mathbb{Z}[y_1, \ldots, y_k] \to \mathbb{Z}[y]$, where

$$y_1 \mapsto y,\ y_2 \mapsto y^{(p+q)d_1},\ \ldots,\ y_k \mapsto y^{(p+q)^{k-1} d_1 \cdots d_{k-1}},$$

allows us to decode $\mathtt{res}(\psi(f), \psi(g)) = \psi(\mathtt{res}(f,g))$ and obtain $\mathtt{res}(f,g)$. The same holds for every polynomial in $\mathbf{SR}(f,g)$. Now $\psi(f), \psi(g) \in (\mathbb{Z}[y])[x]$ have $y-$degree $\leq (p+q)^{k-1} d$ since, in the worst case, f or g hold a monomial such as $y_1^{d_1} y_2^{d_2} \ldots y_k^{d_k}$. Thus, $\mathsf{dg}_y(\mathtt{res}(\psi(f), \psi(g))) < (p+q)^k d$.

PROPOSITION 2.8. *[27] We can compute $\mathbf{SQ}(f,g)$, any polynomial in $\mathbf{SR}(f,g)$, and $\mathtt{res}(f,g)$ in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d\tau)$.*

LEMMA 2.9. *$\mathbf{SR}(f,g)$ is computed in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+2} d\tau)$.*

THEOREM 2.10. *We can evaluate $\mathbf{SR}(f,g)$ at $x = \alpha$, where $\mathsf{a} \in \mathbb{Q} \cup \{\infty\}$ and $\mathcal{L}(a) = \sigma$, in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d \max\{\tau, \sigma\})$.*

PROOF. Compute $\mathbf{SQ}(f,g)$ in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d\,\tau)$ (prop. 2.8), then evaluate it over a, using binary segmentation. For this we need to bound the bitsize of the resulting polynomials.

The polynomials in $\mathbf{SR}(f,g)$ have total degree in $y_1, \ldots, y_k$ bounded by $(p+q)\sum_{i=1}^k d_i$ and coefficient bitsize bounded by $(p+q)\tau$. With respect to x, the polynomials in $\mathbf{SR}(f,g)$ have degrees in $\mathcal{O}(p)$, so substitution $x = \mathsf{a}$ yields values of size $\widetilde{\mathcal{O}}(p\sigma)$. After the evaluation we obtain polynomials in $\mathbb{Z}[y_1, \ldots, y_k]$ with coefficient bitsize bounded by $\max\{(p+q)\tau, p\sigma\} \leq (p+q)\max\{\tau, \sigma\}$.

Consider $\chi : \mathbb{Z}[y] \to \mathbb{Z}$, such that $y \mapsto 2^{\lceil \mathsf{c}(p+q)\max\{\tau,\sigma\}\rceil}$, for a suitable constant c. Apply the map $\phi = \psi \circ \chi$ to f, g. Now, $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leq \mathsf{c} d (p+q)^k \max\{\tau, \sigma\}$. By prop. 2.2, the evaluation costs $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d \max\{\tau, \sigma\})$. □

We obtain the following for $f, g \in (\mathbb{Z}[y])[x]$, such that $\mathsf{dg}_x(f) = p$, $\mathsf{dg}_x(g) = q$, $\mathsf{dg}_y(f), \mathsf{dg}_y(g) \leq d$.

Algorithm 1: SIGN_AT(F, α, β)

Input: $F \in \mathbb{Z}[x,y], \alpha \cong (A, [\mathsf{a}_1, \mathsf{a}_2]), \beta \cong (B, [\mathsf{b}_1, \mathsf{b}_2])$
Output: $\mathrm{sign}(F(\alpha, \beta))$

1 compute $\mathbf{SQ}_x(A, F)$
2 $L_1 \leftarrow \mathbf{SR}_x(A, F\,;\, \mathsf{a}_1)$, $V_1 \leftarrow \emptyset$
3 **foreach** $f \in L_1$ **do** $V_1 \leftarrow \mathrm{ADD}(V_1,\ \mathrm{SIGN\_AT}(f, \beta))$
4 $L_2 \leftarrow SR_x(A, F\,;\, \mathsf{a}_2)$, $V_2 \leftarrow \emptyset$
5 **foreach** $f \in L_2$ **do** $V_2 \leftarrow \mathrm{ADD}(V_2,\ \mathrm{SIGN\_AT}(f, \beta))$
6 RETURN $(\mathrm{VAR}(V_1) - \mathrm{VAR}(V_2)) \cdot \mathrm{sign}(A'(\alpha))$

COROLLARY 2.11. *We compute $\mathbf{SR}(f,g)$ in $\widetilde{\mathcal{O}}_B(pq(p+q)^2 d\tau)$. For any polynomial $\mathbf{SR}_j(f,g)$ in $\mathbf{SR}(f,g)$, $\mathsf{dg}_x(\mathbf{SR}_j(f,g)) = \mathcal{O}(\max\{p,q\})$, $\mathsf{dg}_y(\mathbf{SR}_j(f,g)) = \mathcal{O}(\max\{p,q\}d)$, and also $\mathcal{L}(\mathbf{SR}_j(f,g)) = \mathcal{O}(\max\{p,q\}\tau)$.*

COROLLARY 2.12. *We compute $\mathbf{SQ}(f,g)$, any polynomial in $\mathbf{SR}(f,g)$, and $\mathtt{res}(f,g)$ in $\widetilde{\mathcal{O}}_B(pq \max\{p,q\} d\tau)$.*

COROLLARY 2.13. *We compute $\mathbf{SR}(f,g\,;\,\mathsf{a})$, where $\mathsf{a} \in \mathbb{Q} \cup \{\infty\}$ and $\mathcal{L}(\mathsf{a}) = \sigma$, in $\widetilde{\mathcal{O}}_B(pq \max\{p,q\} d \max\{\tau, \sigma\})$. For the polynomials $\mathbf{SR}_j(f,g\,;\,\mathsf{a}) \in \mathbb{Z}[y]$, except for f, g, we have $\mathsf{dg}_y(\mathbf{SR}_j(f,g\,;\,\mathsf{a})) = \mathcal{O}((p+q)d)$ and $\mathcal{L}(\mathbf{SR}_j(f,g\,;\,\mathsf{a})) = \mathcal{O}(\max\{p,q\}\tau + \min\{p,q\}\sigma)$.*

We now reduce the computation of the sign of $F \in \mathbb{Z}[x,y]$ over $(\alpha, \beta) \in \mathbb{R}^2_{alg}$ to that over several points in $\mathbb{Q}^2$. Let $\mathsf{dg}_x(F) = \mathsf{dg}_y(F) = n_1$, $\mathcal{L}(F) = \sigma$ and $\alpha \cong (A, [\mathsf{a}_1, \mathsf{a}_2])$, $\beta \cong (B, [\mathsf{b}_1, \mathsf{b}_2])$, where $A, B \in \mathbb{Z}[X]$, $\mathsf{dg}(A) = \mathsf{dg}(B) = n_2$, $\mathcal{L}(A) = \mathcal{L}(B) = \sigma$. We assume $n_1 \leq n_2$, which is relevant below. The algorithm is alg. 1, see [30], and generalizes the univariate case, e.g. [10, 36]. For A, resp. B, we assume that we know their values on $\mathsf{a}_1, \mathsf{a}_2$, resp. $\mathsf{b}_1, \mathsf{b}_2$.

THEOREM 2.14 (SIGN_AT). *We compute the sign of polynomial $F(x,y)$ over α, β in $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$.*

PROOF. First, we compute $\mathbf{SQ}_x(A, F)$ so as to evaluate $\mathbf{SR}(A, F)$ on the endpoints of α, in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^2 \sigma)$ (cor. 2.12).

We compute $\mathbf{SR}(A, F; \mathsf{a}_1)$. The first polynomial in the sequence is A, but we already know its value on a_1. This computation costs $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$ by cor. 2.13 with $q = n_1$, $p = n_2$, $d = n_1$, $\tau = \sigma$, and $\sigma = n_2\sigma$, where the latter corresponds to the bitsize of the endpoints. After the evaluation we obtain a list L_1, which contains $\mathcal{O}(n_1)$ polynomials, say $f \in \mathbb{Z}[y]$, such that $\mathsf{dg}(f) = \mathcal{O}(n_1 n_2)$. To bound the bitsize, notice that the polynomials in $\mathbf{SR}(f,g)$ are of degrees $\mathcal{O}(n_1)$ w.r.t. x and of bitsize $\mathcal{O}(n_2\sigma)$. After we evaluate on a_1, $\mathcal{L}(f) = \mathcal{O}(n_1 n_2 \sigma)$.

For each $f \in L_1$ we compute its sign over β and count the sign variations. We could apply directly cor. 2.5, but we can do better. If $\mathsf{dg}(f) \geq n_2$ then $\mathbf{SR}(B, f) = (B, f, -B, g = -\,\mathsf{prem}(f, -B), \ldots)$. We start the evaluations at g: it is computed in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \sigma)$ (prop. 2.1), $\mathsf{dg}(g) = \mathcal{O}(n_2)$ and $\mathcal{L}(g) = \mathcal{O}(n_1 n_2 \sigma)$. Thus, we evaluate $\mathbf{SR}(-B, g; \mathsf{a}_1)$ in $\widetilde{\mathcal{O}}_B(n_1 n_2^3 \sigma)$, by cor. 2.5, with $p = q = n_2$, $\tau_f = \sigma$, $\tau = n_1 n_2 \sigma$. If $\mathsf{dg}(f) < n_2$ the complexity is dominated. Since we perform $\mathcal{O}(n_1)$ such evaluations, all of them cost $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \sigma)$.

We repeat for the other endpoint of α, subtract the sign variations, and multiply by $\mathrm{sign}(A'(\alpha))$, which is known from the process that isolated α. If the last sign in the two sequences is alternating, then $\mathrm{sign}(F(\alpha, \beta)) = 0$. □

3. BIVARIATE REAL SOLVING

Let $F, G \in \mathbb{Z}[x, y]$, $\mathtt{dg}(F) = \mathtt{dg}(G) = n$ and $\mathcal{L}(F) = \mathcal{L}(G) = \sigma$. We assume relatively prime polynomials for simplicity but this hypothesis is not restrictive because it can be verified and if it does not hold, it can be imposed within the same asymptotic complexity. We study algorithms and their complexity for real solving the system $F = G = 0$. The main idea is to project the roots on the x and y axes, to compute the coordinates of the real solutions and somehow to match them. The difference between the algorithms is the way they match solutions.

3.1 The GRID algorithm

Algorithm GRID is straightforward, see also [11, 34]. We compute the $x-$ and $y-$coordinates of the real solutions, as real roots of the resultants $\mathtt{res}_x(F, G)$ and $\mathtt{res}_y(F, G)$. Then, we match them using the algorithm SIGN_AT (th. 2.14) by testing all rectangles in this grid. The output is a list of pairs of real algebraic numbers represented in isolating interval representation. The algorithm also outputs rational axis-aligned rectangles, guaranteed to contain a single root of the system.

To the best of our knowledge, this is the first time that the algorithm's complexity is studied. The disadvantage of the algorithm is that exact implementation of SIGN_AT (alg. 1) is not efficient. However, its simplicity makes it attractive. The algorithm requires no genericity assumption on the input; we study a generic shear that brings the system to generic position in order to compute the multiplicities within the same complexity bound.

The algorithm allows the use of heuristics. In particular, we may exploit easily computed bounds on the number of roots, such as the Mixed Volume or count the roots with a given abscissa α by th. 4.1.

THEOREM 3.1. *Isolating all real roots of system $F = G = 0$ using* GRID *has complexity $\widetilde{\mathcal{O}}_B(n^{14} + n^{13}\sigma)$, provided $\sigma = O(n^3)$.*

PROOF. First we compute the resultant of F and G w.r.t. y, i.e. R_x. The complexity is $\widetilde{\mathcal{O}}_B(n^4\sigma)$, using cor. 2.12. Notice that $\mathtt{dg}(R_x) = \mathcal{O}(n^2)$ and $\mathcal{L}(R_x) = \mathcal{O}(n\,\sigma)$. We isolate its real roots in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$ (prop. 2.4) and store them in L_x. This complexity shall be dominated. We do the same for the y axis and store the roots in L_y.

The representation of the real algebraic numbers contains the square-free part of R_x, or R_y. In both cases the bitsize of the polynomial is $\mathcal{O}(n^2 + n\,\sigma)$ [3, 10]. The isolating intervals have endpoints of size $\mathcal{O}(n^4 + n^3\,\sigma)$.

Let r_x, resp. r_y be the number of real roots of the corresponding resultants. Both are bounded by $\mathcal{O}(n^2)$. We form all possible pairs of real algebraic numbers from L_x and L_y and check for every such pair if both F and G vanish, using SIGN_AT (th. 2.14). Each evaluation costs $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$ and we perform $r_x\, r_y = O(n^4)$ of them. □

We now examine the multiplicity of a root (α, β) of the system. Refer to [4, sec.II.6] for its definition as the exponent of factor $(\beta x - \alpha y)$ in the resultant of the (homogenized) polynomials, under certain assumptions.

The algorithm reduces to bivariate sign determination and does not require bivariate factorization. We shall use the resultant, since it allows for multiplicities to "project". Previous work includes [12, 31, 35]. The sum of multiplicities of all roots (α, β_j) equals the multiplicity of $x = \alpha$ in the respective resultant. It is possible to apply a shear transform to the coordinate frame so as to ensure that different roots project to different points on the x-axis. We determine an adequate (horizontal) shear such that

$$R_t(x) = \mathtt{res}_y\left(F(x + ty, y), G(x + ty, y)\right), \qquad (1)$$

when $t \mapsto t_0 \in \mathbb{Z}$, has simple roots corresponding to the projections of the common roots of the system $F(x, y) = G(x, y) = 0$ and the degree of the polynomials remains the same. Notice that this shear does not affect inherently multiple roots, which exist independently of the reference frame. $R_{red} \in (\mathbb{Z}[t])[x]$ is the squarefree part of the resultant, as an element of UFD $(\mathbb{Z}[t])[x]$, and its discriminant, with respect to x, is $\Delta \in \mathbb{Z}[t]$. Then t_0 must be such that $\Delta(t_0) \neq 0$.

LEMMA 3.2. *Computing $t_0 \in \mathbb{Z}$, such that the corresponding shear is sufficiently generic, has complexity $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$.*

The idea here is to use explicit candidate values of t_0 right from the start [7]. In practice, the above complexity becomes $\widetilde{\mathcal{O}}_B(n^5\sigma)$, because a constant number of tries or a random value will typically suffice. For an alternative approach see [14], also [3]. It is straightforward to compute the multiplicities of the sheared system. Then, we need to match the latter with the roots of the original system, which is nontrivial in practice.

THEOREM 3.3. *Consider the setting of th. 3.1. Having isolated all real roots of $F = G = 0$, it is possible to determine their multiplicities in $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma^2)$.*

Algorithm 2: M_RUR (F, G)

Input: $F, G \in \mathbb{Z}[X, Y]$ in generic position
Output: The real solutions of the system $F = G = 0$

1 $\mathbf{SR} \leftarrow \mathbf{SR}_y(F, G)$
/* Projections and real solving with multiplicities */
2 $R_x \leftarrow \mathtt{res}_y(F, G)$
3 $P_x, M_x \leftarrow$ SOLVE(R_x)
4 $R_y \leftarrow \mathtt{res}_x(F, G)$
5 $P_y, M_y \leftarrow$ SOLVE(R_y)
6 $I \leftarrow$ INTERMEDIATE_POINTS(P_y)
/* Factorization of R_x according to sr */
7 $K \leftarrow$ COMPUTE_K$(\mathbf{SR}, P_x)$
8 $Q \leftarrow \emptyset$
/* Matching the solutions */
9 **foreach** $\alpha \in P_x$ **do**
10 $\quad \beta \leftarrow$ FIND(α, K, P_y, I)
11 $\quad Q \leftarrow$ ADD$(Q, \{\alpha, \beta\})$
12 RETURN Q

3.2 The M_RUR algorithm

M_RUR assumes that the polynomials are in Generic Position: different roots project to different x-coordinates and leading coefficients w.r.t. y have no common real roots.

PROPOSITION 3.4. *[12, 3] Let F, G be co-prime polynomials, in generic position. If $\mathbf{SR}_j(x, y) = \mathbf{sr}_j(x)y^j + \mathbf{sr}_{j,j-1}(x)y^{j-1} + \cdots + \mathbf{sr}_{j,0}(x)$, and (α, β) is a real solution of the system $F = G = 0$, then there exists k, such that $\mathbf{sr}_0(\alpha) = \cdots = \mathbf{sr}_{k-1}(\alpha) = 0$, $\mathbf{sr}_k(\alpha) \neq 0$ and $\beta = -\frac{1}{k}\frac{\mathbf{sr}_{k,k-1}(\alpha)}{\mathbf{sr}_k(\alpha)}$.*

This expresses the ordinate of a solution in a Rational Univariate Representation (RUR) of the abscissa. The RUR applies to multivariate algebraic systems [28, 5, 29, 3]; it generalizes the primitive element method by Kronecker. Here we adapt it to small-dimensional systems.

Our algorithm is similar to [14, 12]. However, their algorithm computes only a RUR using prop. 3.4, so the representation of the ordinates remains implicit. Often, this representation is not sufficient (we can always compute the minimal polynomial of the roots, but this is highly inefficient). We modified the algorithm [11], so that the output includes isolating rectangles, hence the name modified-RUR (M_RUR). The most important difference with [12] is that they represent algebraic numbers by Thom's encoding while we use isolating intervals, which were thought of having high theoretical complexity. We will prove that this is not the case.

The pseudo-code of M_RUR is in alg. 2. We project on the x and the y-axis; for each real solution on the x-axis we compute its ordinate using prop. 3.4. First we compute the sequence $\mathbf{SR}(F,G)$ w.r.t. y in $\widetilde{\mathcal{O}}_B(n^5\,\sigma)$ (cor. 2.11).

Projection. This is similar to GRID. The complexity is dominated by real solving the resultants, i.e. $\widetilde{\mathcal{O}}_B(n^{12}+n^{10}\,\sigma^2)$. Let α_i, resp. β_j, be the real root coordinates. We compute rationals q_j between the β_j's in $\widetilde{\mathcal{O}}_B(n^5\sigma)$, viz. INTERMEDIATE_POINTS(P_y); the q_j have aggregate bitsize $\mathcal{O}(n^3\,\sigma)$:

$$q_0 < \beta_1 < q_1 < \beta_2 < \cdots < \beta_{\ell-1} < q_{\ell-1} < \beta_\ell < q_\ell, \qquad (2)$$

where $\ell \leq 2\,n^2$. Every β_j corresponds to a unique α_i. The multiplicity of α_i as a root of R_x is the multiplicity of a real solution of the system, that has it as abscissa.

Sub-algorithm COMPUTE_K. In order to apply prop. 3.4, for every α_i we must compute $k \in \mathbb{N}^*$ such the assumptions of the theorem are fulfilled; this is possible by genericity. We follow [23, 12] and define recursively polynomials $\Gamma_j(x)$: Let $\Phi_0(x) = \frac{\mathbf{sr}_0(x)}{\mathsf{gcd}(\mathbf{sr}_0(x),\mathbf{sr}_0'(x))}$, $\Phi_j(x) = \mathsf{gcd}(\Phi_{j-1}(x), \mathbf{sr}_j(x))$, and $\Gamma_j = \frac{\Phi_{j-1}(x)}{\Phi_j(x)}$, for $j>0$. Now $\mathbf{sr}_i(x) \in \mathbb{Z}[x]$ is the principal subresultant coefficient of $\mathbf{SR}_i \in (\mathbb{Z}[x])[y]$, and $\Phi_0(x)$ is the square-free part of $R_x = \mathbf{sr}_0(x)$. By construction, $\Phi_0(x) = \prod_j \Gamma_j(x)$ and $\mathsf{gcd}(\Gamma_j,\Gamma_i)=1$, if $j\neq i$. Hence every α_i is a root of a unique Γ_j and the latter switches sign at the interval's endpoints. Then, $\mathbf{sr}_0(\alpha)=\mathbf{sr}_1(\alpha)=0,\ldots,\mathbf{sr}_j(\alpha)=0$, $\mathbf{sr}_{j+1}(\alpha)\neq 0$; thus $k=j+1$.

It holds that $\mathsf{dg}(\Phi_0)=\mathcal{O}(n^2)$ and $\mathcal{L}(\Phi_0)=\mathcal{O}(n^2+n\,\sigma)$. Moreover, $\sum_j \mathsf{dg}(\Gamma_j)=\sum_j \delta_j = \mathcal{O}(n^2)$ and, by Mignotte's bound [20], $\mathcal{L}(\Gamma_j)=\mathcal{O}(n^2+n\sigma)$. To compute the factorization $\Phi_0(x)=\prod_j \Gamma_j(x)$ as a product of the $\mathbf{sr}_j(x)$, we perform $\mathcal{O}(n)$ gcd computations of polynomials of degree $\mathcal{O}(n^2)$ and bitsize $\widetilde{\mathcal{O}}(n^2+n\sigma)$. Each gcd computation costs $\widetilde{\mathcal{O}}_B(n^6+n^5\,\sigma)$ (prop. 2.1) and thus the overall cost is $\widetilde{\mathcal{O}}_B(n^7+n^6\,\sigma)$.

We compute the sign of the Γ_j over all the $O(n^2)$ isolating endpoints of the α_i, which have aggregate bitsize $\mathcal{O}(n^4+n^3\,\sigma)$ (lem. 2.6) in $\widetilde{\mathcal{O}}_B(\delta_j n^4+\delta_j n^3\sigma+\delta_j^2(n^4+n^3\sigma))$, using Horner's rule. Summing over all δ_j, the complexity is $\widetilde{\mathcal{O}}_B(n^8+n^7\sigma)$. Thus the overall complexity is $\widetilde{\mathcal{O}}_B(n^9+n^8\,\sigma)$.

Matching and algorithm FIND. The process takes a real root of R_x and computes ordinate β of the corresponding root of the system. For some real root α of R_x we represent the ordinate $A(\alpha) = -\frac{1}{k}\frac{\mathbf{sr}_{k,k-1}(\alpha)}{\mathbf{sr}_k(\alpha)} = \frac{A_1(\alpha)}{A_2(\alpha)}$. The generic position assumption guarantees that there is a unique β_j, in P_y, such that $\beta_j = A(\alpha)$, where $1\leq j\leq \ell$. In order to compute j we use (2): $q_j < A(\alpha) = \frac{A_1(\alpha)}{A_2(\alpha)} = \beta_j < q_{j+1}$. Thus j can be computed by binary search in $\mathcal{O}(\lg\ell)=\mathcal{O}(\lg n)$ comparisons of $A(\alpha)$ with the q_j. This is equivalent to computing the sign of $B_j(X) = A_1(X) - q_j\,A_2(X)$ over α by executing $\mathcal{O}(\lg n)$ times, SIGN_AT(B_j,α).

Now, $\mathcal{L}(q_j)=\mathcal{O}(n^4+n^3\sigma)$ and $\mathsf{dg}(A_1)=\mathsf{dg}(\mathbf{sr}_{k,k-1})=\mathcal{O}(n^2)$, $\mathsf{dg}(A_2)=\mathsf{dg}(\mathbf{sr}_k)=\mathcal{O}(n^2)$, $\mathcal{L}(A_1)=\mathcal{O}(n\,\sigma)$, $\mathcal{L}(A_2)=\mathcal{O}(n\,\sigma)$. Thus $\mathsf{dg}(B_j)=\mathcal{O}(n^2)$ and $\mathcal{L}(B_j)=\mathcal{O}(n^4+n^3\,\sigma)$. We conclude that SIGN_AT(B_j,α) and FIND have complexity $\widetilde{\mathcal{O}}_B(n^8+n^7\sigma)$ (cor. 2.5). As for the overall complexity of the loop (Lines 9-11) the complexity is $\widetilde{\mathcal{O}}_B(n^{10}+n^9\sigma)$, since it is executed $\mathcal{O}(n^2)$ times.

THEOREM 3.5. *We isolate all real roots of $F=G=0$, if F, G are in generic position, by* M_RUR *in* $\widetilde{\mathcal{O}}_B(n^{12}+n^{10}\sigma)$.

The generic position assumption is without loss of generality since we can always put the system in such position by applying a shear transform; see previous section. The bitsize of polynomials of the (sheared) system becomes $\widetilde{\mathcal{O}}(n+\sigma)$ [12] and does not change the bound of th. 3.5. However, now is raised the problem of expressing the real roots in the original coordinate system (see also the proof of th. 3.3).

3.3 The G_RUR algorithm

We present an algorithm that uses some ideas from RUR but relies on GCD computations of polynomials with coefficients in an extension field to achieve efficiency (hence the name G_RUR). For the GCD computations we use the algorithm (and the implementation) of [32].

The first steps are similar to the previous algorithms: We project on the axes, we perform real solving and compute the intermediate points on the y-axis. The complexity is $\widetilde{\mathcal{O}}_B(n^{12}+n^{10}\sigma^2)$.

For each x-coordinate, say α, we compute the square-free part of $F(\alpha,y)$ and $G(\alpha,y)$, say $\bar{F}$ and $\bar{G}$. The complexity is that of computing the gcd with the derivative. In [32] the cost is $\widetilde{\mathcal{O}}_B(mMND+mN^2D^2+m^2kD)$, where M is the bitsize of the largest coefficient, N is the degree of the largest polynomial, D is the degree of the extension, k is the degree of the gcd, and m is the number of primes needed. The complexity does not assume fast multiplication algorithms, thus, under this assumption, it becomes $\widetilde{\mathcal{O}}_B(mMND+mND+mkD)$.

In our case $M=\mathcal{O}(\sigma)$, $N=\mathcal{O}(n)$, $D=\mathcal{O}(n^2)$, $k=\mathcal{O}(n)$, and $m=\mathcal{O}(n\sigma)$. The cost is $\widetilde{\mathcal{O}}_B(n^4\sigma^2)$ and since we have to do it $\mathcal{O}(n^2)$ times, the overall cost is $\widetilde{\mathcal{O}}_B(n^6\sigma^2)$. Notice the bitsize of the result is $\widetilde{\mathcal{O}}_B(n+\sigma)$ [3].

Now for each α, we compute $H=\gcd(\bar{F},\bar{G})$. We have $M=\mathcal{O}(n+\sigma)$, $N=\mathcal{O}(n)$, $D=\mathcal{O}(n^2)$, $k=\mathcal{O}(n)$, and $m=\mathcal{O}(n^2+n\sigma)$ and so the cost of each operation is $\widetilde{\mathcal{O}}_B(n^6+n^4\sigma^2)$ and overall $\widetilde{\mathcal{O}}_B(n^8+n^6\sigma^2)$. The size of m comes from Mignotte's bound [20]. Notice that H is a square-free polynomial in $(\mathbb{Z}[\alpha])[y]$, of degree $\mathcal{O}(n)$ and bitsize $\mathcal{O}(n^2+n\sigma)$, the real roots of which correspond to the real solutions of the system with abscissa α. It should change sign only over the intervals that contain its real roots. To check these signs, we have to substitute y in H by the intermediate points, thus obtaining a polynomial in $\mathbb{Z}[\alpha]$, of degree $\mathcal{O}(n)$ and bitsize $\mathcal{O}(n^2+n\sigma+ns_j)$, where s_j is the bitsize of the

j-th intermediate point.

Now, we consider this polynomial in $\mathbb{Z}[x]$ and evaluate it over α. Using cor. 2.5 with $p = n^2$, $\tau_f = n^2 + n\sigma$, $q = n$, and $\tau_g = n^2 + n\sigma + ns_j$, this costs $\widetilde{\mathcal{O}}_B(n^6 + n^5\sigma + n^4 s_j)$. Summing over $\mathcal{O}(n^2)$ points and using lem. 2.6, we obtain $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$. Thus, the overall complexity is $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$.

THEOREM 3.6. *We can isolate the real roots of the system $F = G = 0$, using* G_RUR *in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma)$.*

4. APPLICATIONS

Real root counting. We wish to count the number of roots of $\bar{F} = F(\alpha, y) \in (\mathbb{Z}[\alpha])[y]$ in $\mathbb{R}$, in $(\mathsf{c}, +\infty)$ and in $(\beta, +\infty)$. Assume $\alpha, \beta \in \mathbb{R}_{alg}$ as above, but with $\mathcal{L}(A), \mathcal{L}(B) \leq \tau$ and $\mathsf{c} \in \mathbb{Q}$, such that $\mathcal{L}(\mathsf{c}) = \lambda$. Moreover, let $n_1^2 = \mathcal{O}(n_2)$, as will be the case in applications.

THEOREM 4.1. *We count the real roots of $\bar{F}$ in $(-\infty, +\infty)$, $(\beta, +\infty)$ and $(\mathsf{c}, +\infty)$, respectively, in $\widetilde{\mathcal{O}}_B(n_1^4 n_2\sigma + n_1^5 n_2\tau)$, $\widetilde{\mathcal{O}}_B(n_1^5 n_2^3 \max\{n_1\sigma, \tau\})$ and $\widetilde{\mathcal{O}}_B(n_1^4 n_2 \max\{n_1\tau, \sigma, \lambda\})$.*

The proof uses Sturm's theorem and the good specialization properties of subresultants in order to switch the order of substitution $x = \alpha$ and sequence computation; see [7].

Simultaneous inequalities in two variables. Let P, Q, $A_1, \ldots, A_{\ell_1}$, $B_1, \ldots, B_{\ell_2}$, $C_1, \ldots, C_{\ell_3} \in \mathbb{Z}[X, Y]$, such that their total degrees are bounded by n and their bitsize by σ. We wish to compute $(\alpha, \beta) \in \mathbb{R}_{alg}^2$ such that $P(\alpha, \beta) = Q(\alpha, \beta) = 0$ and also $A_i(\alpha, \beta) > 0$, $B_j(\alpha, \beta) < 0$ and $C_k(\alpha, \beta) = 0$, where $1 \leq i \leq \ell_1, 1 \leq j \leq \ell_2, 1 \leq k \leq \ell_3$. Let $\ell = \ell_1 + \ell_2 + \ell_3$.

COROLLARY 4.2. *There is an algorithm that solves the problem of ℓ simultaneous inequalities of degree $\leq n$ and bitsize $\leq \sigma$, in $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell n^{11}\sigma + n^{10}\sigma^2)$.*

The complexity of topology. We improve the complexity of computing the topology of a real plane algebraic curve. See [3, 12, 23] for the algorithm.

We consider the curve, in generic position, defined by $F \in \mathbb{Z}[x, y]$, such that $\mathtt{dg}(F) = n$ and $\mathcal{L}(F) = \sigma$. We compute the critical points of the curve, i.e. solve $F = F_y = 0$ in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$. Next, we compute the intermediate points on the x axis, in $\widetilde{\mathcal{O}}_B(n^4 + n^3\sigma)$ (lem. 2.6). For each intermediate point, say q_j, we need to compute the number of branches of the curve that cross the vertical line $x = q_j$. This is equivalent to computing the number of real solutions of the polynomial $F(q_j, y) \in \mathbb{Z}[y]$, which has degree d and bitsize $\mathcal{O}(n\mathcal{L}(q_j))$. For this we use Sturm's theorem and th. 2.2 and the cost is $\widetilde{\mathcal{O}}_B(n^3\mathcal{L}(q_j))$. For all q_j's the cost is $\widetilde{\mathcal{O}}_B(n^7 + n^6\sigma)$.

For each critical point, say (α, β) we need to compute the number of branches of the curve that cross the vertical line $x = \alpha$, and the number of them that are above $y = \beta$. The first task corresponds to computing the number of real roots of $F(\alpha, y)$, by application of th. 4.1, in $\widetilde{\mathcal{O}}_B(n^9 + n^8\sigma)$, where $n_1 = n$, $n_2 = n^2$, and $\tau = n^2 + n\sigma$. Since there are $\mathcal{O}(n^2)$ critical values, the overall cost of the step is $\widetilde{\mathcal{O}}_B(n^{11} + n^{10}\sigma)$.

Finally, we compute the number of branches that cross the line $x = \alpha$ and are above $y = \beta$. We do this by th. 4.1, in $\widetilde{\mathcal{O}}_B(n^{13} + n^{12}\sigma)$. Since there are $\mathcal{O}(n^2)$ critical points, the complexity is $\widetilde{\mathcal{O}}_B(n^{15} + n^{14}\sigma)$. It remains to connect the critical points according to the information that we have for the branches. The complexity of this step is dominated. It now follows that the complexity of the algorithm is $\widetilde{\mathcal{O}}_B(n^{15} + n^{14}\sigma + n^{10}\sigma^2)$, or $\widetilde{\mathcal{O}}_B(N^{15})$, which is worse by a factor than [3].

We improve the complexity of the last step since M_RUR computes the RUR representation of the ordinates. Thus, instead of performing bivariate sign evaluations in order to compute the number of branches above $y = \beta$, we can substitute the RUR representation of β and perform univariate sign evaluations. This corresponds to computing the sign of $\mathcal{O}(n^2)$ polynomials of degree $\mathcal{O}(n^2)$ and bitsize $\mathcal{O}(n^4 + n^3\sigma)$, over all the α's [12]. Using lem. 2.7 for each polynomial the cost is $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$, and since there are $\widetilde{\mathcal{O}}_B(n^2)$ of them, the total cost is $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma)$.

THEOREM 4.3. *We compute the topology of a real plane algebraic curve, defined by a polynomial of degree n and bitsize σ, in $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma)$.*

Thus the overall complexity of the algorithm improves the previously known bound by a factor of N^2. We assumed generic position, since we can apply a shear to achieve this, see sec. 3.1.

5. IMPLEMENTATION AND EXPERIMENTS

We describe our open source MAPLE implementation[1] and illustrate its capabilities through comparative experiments. The design is based on object oriented programming and the generic programming paradigm in view of transferring our implementation to `C++`.

The class of real algebraic numbers represents them in isolating interval representations. We provide various algorithms for computing signed polynomial remainder sequences; real solving univariate polynomials using Sturm's algorithm; computations with one and two real algebraic numbers, such as sign evaluation, comparison; and our algorithms for real solving of bivariate systems. Computations are performed first using intervals with floating point arithmetic and, if they fail, then an exact algorithm using rational arithmetic is called. For GCD computations in an extension field we use the MAPLE package of [32]. We have not implemented, yet, the optimal algorithms for computing and evaluating polynomial remainder sequences.

Overall performance results are shown on tab. 1, averaged over 10 iterations. Systems R_i, M_i, and D_i are presented in [11], systems C_i in [14], and W_i are the C_i after swapping the x and y variables. For the first data set, there are no timings for INSULATE and TOP since it was not easy to modify their code so as to deal with general polynomial systems. The rest correspond to algebraic curves, i.e. polynomial systems of the form $f = f_y = 0$, that all packages can deal with.

It seems that G_RUR is our solver of choice since it is faster than GRID and M_RUR in 17 out of our 18 instances. However, this may not hold when the extension field is of high degree. G_RUR yields solutions in less than a second, apart from system C_5. Overall, for total degrees ≤ 8, G_RUR requires less than 0.4 secs to respond. As a result, G_RUR is 7-11 times faster than GRID, and about 38 times than M_RUR.

[1] `www.di.uoa.gr/~erga/soft/SLV_index.html`

system	deg		solutions	Average Time (msecs)									
				BIVARIATE SOLVING							TOPOLOGY		
				this paper (SLV)			FGb/Rs	Synaps			Insulate	Top	
	f	g		grid	m_rur	g_rur		sturm	subdiv	newmac		60	500
R_1	3	4	2	5	9	5	26	2	2	5	–	–	–
R_2	3	1	1	66	21	36	24	1	1	1	–	–	–
R_3	3	1	1	1	2	1	22	1	2	1	–	–	–
M_1	3	3	4	87	72	10	25	2	1	2	–	–	–
M_2	4	2	3	4	5	4	24	1	289	2	–	–	–
M_3	6	3	5	803	782	110	30	230	5,058	7	–	–	–
M_4	9	10	2	218	389	210	158	90	3	447	–	–	–
D_1	4	5	1	6	12	6	28	2	5	8	–	–	–
D_2	2	2	4	667	147	128	26	21	1	2	–	–	–
C_1	7	6	6	1,896	954	222	93	479	170,265	39	524	409	1,367
C_2	4	3	6	177	234	18	27	12	23	4	28	36	115
C_3	8	7	13	580	1,815	75	54	23	214	25	327	693	2,829
C_4	8	7	17	5,903	80,650	370	138	3,495	217	190	1,589	1,624	6,435
C_5	16	15	17	$> 20'$	60,832	3,877	4,044	$> 20'$	6,345	346	179,182	91,993	180,917
W_1	7	6	9	2,293	2,115	247	92	954	55,040	39	517	419	1,350
W_2	4	3	5	367	283	114	29	20	224	3	27	20	60
W_3	8	7	13	518	2,333	24	56	32	285	25	309	525	1,588
W_4	8	7	17	5,410	77,207	280	148	4,086	280	207	1,579	1,458	4,830

Table 1: Performance averages over 10 runs in maple 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

One reason is that the sheared systems that M_RUR solves are dense and of increased bitsize.

Among our algorithms, GRID and M_RUR benefit the most from filtering. G_RUR gains only a factor of 1.1-2. GRID gains a factor of 2-5. In M_RUR we use one more filtering heuristic technique: after computing the intermediate points on the y-axis, we perform refining with [1] (up to 20 times on systems with high degree) on the intervals of the candidate solutions along the x-axis. Recall that M_RUR binary-searches for solutions along the y-axis. Of course the refinement must not be excessive since this will increase the bitsize of the coefficients and as a result lead to costlier operations in case where filtering techniques fail. This has been very efficient in practice, resulting on average an additional speedup of 2.2-3.4; overall filtering improves M_RUR by a factor of 7-11.

If a polynomial system did not comply with the generic position criterion required by M_RUR, we deterministically tested a value for the required shear; in all cases our first candidate ($t = 3$) worked. This is relatively inexpensive on systems with polynomials of degree ≤ 5. For systems with polynomials of higher degree, in some cases the deterministic shear computation is more expensive than real solving. Hence, a random shear is more efficient in general, as suggested also by the asymptotic analysis.

We tested FGB/RS[2] [29], which performs exact real solving using Gröbner bases and RUR, through its MAPLE interface. It should be underlined that communication with MAPLE increases the runtimes. G_RUR is faster in 8 out of the 18 instances, including the difficult system C_5. Lastly, we examined 3 SYNAPS[3] solvers: STURM is a naive implementation of GRID [11]; SUBDIV implements [22], and is based on Bernstein basis and `double` arithmetic. It needs an intial box for computing the real solutions of the system and in all the cases we used $[-10, 10] \times [-10, 10]$. NEWMAC [24], is a general purpose solver based on computations of generalized eigenvectors using LAPACK, which computes all complex solutions. STURM is faster than our MAPLE implementation. SUBDIV and NEWMAC are in general fast solvers.

We also tested other MAPLE implementations: INSULATE is a package that implements [35] for computing the topology of real algebraic curves, and TOP implements [14]. We tried to modify the packages so as to stop them as soon as they compute the real solutions of the corresponding bivariate system. Both packages were kindly provided to us by their authors. TOP has an additional parameter that sets the initial precision (digits). A very low initial precision or a very high one results in inaccuracy or performance loss; but there is no easy way for choosing a good value. Hence, we followed [15] and recorded its performance on initial values of 60 and 500 digits. Compared to G_RUR, INSULATE is 2-46 times slower when the total degree is ≥ 6. On the other hand, TOP is slower than G_RUR 1.7-23 times when the total degree is ≥ 6 and the curves have many critical points.

We underline that we do not consider experiments as competition, but a crucial step for improving existing software. Moreover, it is very difficult to compare different packages, since in most cases they are made for different needs. In addition, accurate timing in MAPLE is hard, since it is a general purpose package and a lot of overhead is added to its function calls. For example this is the case for FGB/RS.

Our implementations and INSULATE have demonstrated the most robust behaviour, not only by replying within our specified time limits, but also no errors were generated during their execution.[4] In the case of FGB/RS, some errors regarding the communication of the application with the MAPLE kernel occurred; i.e. successive calls of the RS_ISOLATE

[2] `http://www-spaces.lip6.fr/index.html`

[3] `http://www-sop.inria.fr/galaad/logiciels/synaps/`

[4] Sto issac version prepei na mpei to keimeno pou den einai gia journal.

	phase of the algorithm	interval		median	mean	std dev
		min	max			
GRID	projections	00.00	00.53	00.04	00.08	00.13
	univ. solving	02.05	99.75	07.08	26.77	35.88
	biv. solving	00.19	97.93	96.18	73.03	36.04
	sorting	00.00	01.13	00.06	00.12	00.26
MRUR	projection	00.00	00.75	00.06	00.14	00.23
	univ. solving	00.18	91.37	15.55	17.47	20.79
	StHa seq.	00.08	38.23	01.17	05.80	09.91
	inter. points	00.00	03.23	00.09	00.32	00.75
	filter x-cand	00.68	72.84	26.68	23.81	21.93
	compute K	00.09	34.37	02.04	07.06	10.21
	biv. solving	01.77	98.32	51.17	45.41	28.71
GRUR	projections	00.02	03.89	00.23	00.48	00.88
	univ. solving	07.99	99.37	39.83	41.68	25.52
	inter. points	00.02	03.81	00.54	01.11	01.28
	rational biv.	00.07	57.07	14.83	15.89	19.81
	$\mathbb{R}_{alg}$ biv.	00.00	91.72	65.30	40.53	36.89
	sorting	00.00	01.50	00.22	00.32	00.43

Table 2: Statistics on the performance from [7].

function in a FOR loop, especially on the difficult system C_5. STURM failed to reply within our time limits for C_5. As for NEWMAC and SUBDIV, some numerical errors are introduced since the former is based on LAPACK and the latter on floating point arithmetic. These solvers fail to compute the correct number of real solutions in many cases. Finally, STURM' s inefficiency, in some experiments, is basically due to the lack of modular algorithms for computing resultants.

GRID and M_RUR demonstrate a high fluctuation in runtimes, compared, e.g. to the stability of NEWMAC or FGB/RS. The latter spends a lot of time on Gröbner bases. The rest of the solvers demonstrate a similar fluctuation, especially those that are based on MAPLE.

To summarize, we believe that the implementation of our algorithms give very encouraging results, at least for polynomial systems of moderate degree.

The time that each algorithm spends on the various steps is on tab. 2 as percentages of the overall computing times in tab. 1. `Projections` shows the time for the computation of the resultants, `Univ. Solving` for real solving the resultants, and `Sorting` for sorting solutions. In GRID's and M_RUR's case, `biv. solving` corresponds to matching. In G_RUR's case timings for matching are divided between `rational biv.` and $\mathbb{R}_{alg}$ `biv.`; the first refers to when at least one of the co-ordinates is a rational number, while the latter indicates timings when both co-ordinates are not rational. `Inter. points` refers to computation of the intermediate points between resultant roots along the y-axis. `StHa seq.` refers to the computation of the **StHa** sequence. `Filter` x`-cand` shows the time for additional filtering. `Compute` K reflects the time for sub-algorithm COMPUTE-K.

In a nutshell, GRID spends more than 73% of its time in matching. Recall that this percent includes the application of filters. M_RUR spends about 45-50% of its time in matching and about 24-27% in the pre-computation filtering technique. G_RUR spends 55-80% of its time in matching, including gcd computations in an extension field.

Acknowledgements. We acknowledge discussions with B. Mourrain, thank J-P. Pavone for his help with SYNAPS, and the anonymous referees for their comments. The third author thanks F. Rouillier for various comments.

6. REFERENCES

[1] J. Abbott. Quadratic interval refinement for real roots. In *ISSAC 2006, poster presentation*. http://www.dima.unige.it/ abbott/.

[2] D. Arnon and S. McCallum. A polynomial time algorithm for the topological type of a real algebraic curve. *JSC*, 5:213–236, 1988.

[3] S. Basu, R. Pollack, and M-F.Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2nd edition, 2006.

[4] E. Brieskorn and H. Knörrer. *Plane Algebraic Curves*. Birkhäuser, Basel, 1986.

[5] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. STOC*, pages 460–467, 1988.

[6] F. Cazals, J.-C. Faugère, M. Pouget, and F. Rouillier. The implicit structure of ridges of a smooth parametric surface. *Comput. Aided Geom. Des.*, 23(7):582–598, 2006.

[7] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. Research Report 6116, INRIA, 02 2007. https://hal.inria.fr/inria-00129309.

[8] Z. Du, V. Sharma, and C. K. Yap. Amortized bound for root isolation via Sturm sequences. In D. Wang and L. Zhi, editors, *Int. Workshop on Symbolic Numeric Computing*, pages 81–93, Beijing, China, 2005.

[9] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the descartes method. In *ISSAC*, pages 71–78, New York, NY, USA, 2006. ACM Press.

[10] I. Z. Emiris, B. Mourrain, and E. P. Tsigaridas. Real Algebraic Numbers: Complexity Analysis and Experimentation. In P. Hertling, C. Hoffmann, W. Luther, and N. Revol, editors, *Reliable Implementations of Real Number Algorithms: Theory and Practice*, LNCS (to appear). Springer Verlag, 2007. also available in www.inria.fr/rrrt/rr-5897.html.

[11] I. Z. Emiris and E. P. Tsigaridas. Real solving of bivariate polynomial systems. In V. Ganzha and E. Mayr, editors, *Proc. Computer Algebra in Scientific Computing (CASC)*, volume 3718 of *LNCS*, pages 150–161. Springer, 2005.

[12] L. González-Vega and M. El Kahoui. An improved upper complexity bound for the topology computation of a real algebraic plane curve. *J. Complexity*, 12(4):527–544, 1996.

[13] L. González-Vega, H. Lombardi, T. Recio, and M.-F. Roy. Sturm-Habicht Sequence. In *ISSAC*, pages 136–146, 1989.

[14] L. Gonzalez-Vega and I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Computer Aided Geometric Design*, 19(9):719–743, Dec. 2002.

[15] M. Kerber. Analysis of real algebraic plane curves. Diploma thesis, MPI Saarbrücken, 2006.

[16] J. Klose. Binary segmentation for multivariate polynomials. *J. Complexity*, 11(3):330–343, 1995.

[17] K. Ko, T. Sakkalis, and N. Patrikalakis. Resolution of multiple roots of nonlinear polynomial systems. *Int. J. of Shape Modeling*, 11(1):121–147, 2005.

[18] T. Lickteig and M.-F. Roy. Sylvester-Habicht Sequences and Fast Cauchy Index Computation. *JSC*, 31(3):315–341, 2001.

[19] H. Lombardi, M.-F. Roy, and M. Safey El Din. New Structure Theorem for Subresultants. *JSC*, 29(4-5):663–689, 2000.

[20] M. Mignotte and D. Stefanescu. *Polynomials: An algorithmic approach*. Springer, 1999.

[21] P. Milne. On the solution of a set of polynomial equations. In B. Donald, D. Kapur, and J. Mundy, editors, *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102. Academic Press, 1992.

[22] B. Mourrain and J.-P. Pavone. Subdivision methods for solving polynomial equations. Technical Report RR-5658, INRIA Sophia-Antipolis, 2005.

[23] B. Mourrain, S. Pion, S. Schmitt, J.-P. Técourt, E. Tsigaridas, and N. Wolpert. Algebraic issues in computational geometry. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 117–155. Springer-Verlag, 2006.

[24] B. Mourrain and P. Trébuchet. Solving projective complete intersection faster. In C. Traverso, editor, *Proc. Intern. Symp. on Symbolic and Algebraic Computation*, pages 231–238. New-York, ACM Press., 2000.

[25] V. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *JSC*, 33(5):701–733, 2002.

[26] P. Pedersen, M.-F. Roy, and A. Szpirglas. Counting real zeros in the multivariate case. In F. Eyssette and A. Galligo, editors, *Computational Algebraic Geometry*, volume 109 of *Progress in Mathematics*, pages 203–224. Birkhäuser, Boston, 1993.

[27] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC*, pages 233–240, 1997.

[28] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of systems of polynomials. *SIAM J. Computing*, 18:350–370, 1989.

[29] F. Rouillier. Solving zero-dimensional systems through the rational univariate representation. *J. of AAECC*, 9(5):433–461, 1999.

[30] T. Sakkalis. Signs of algebraic numbers. *Computers and Mathematics*, pages 131–134, 1989.

[31] T. Sakkalis and R. Farouki. Singular points of algebraic curves. *JSC*, 9(4):405–421, 1990.

[32] M. van Hoeij and M. Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *ISSAC*, pages 109–116, July 2002.

[33] J. von zur Gathen and T. Lücking. Subresultants revisited. *TCS*, 1-3(297):199–239, 2003.

[34] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. PhD thesis, MPI fuer Informatik, Oct. 2002.

[35] N. Wolpert and R. Seidel. On the Exact Computation of the Topology of Real Algebraic Curves. In *SoCG*, pages 107–115. ACM, 2005.

[36] C. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.

APPENDIX

A. ALGORITHMS AND COMPLEXITY

A.1 Univariate Polynomials

PROOF OF LEM. 2.2. Let $\mathbf{SR}_{q+1} = f$ and $\mathbf{SR}_q = g$. For the moment we forget $\mathbf{SR}_{q+1}$. We may assume that $\mathbf{SR}_{q-1}$ is computed, since the cost of computing one element of $\mathbf{SR}$ is the same as that of computing $\mathbf{SQ}(f, g)$ (Pr. 2.1).

We follow Lickteig and Roy [18]. For two polynomials A, B of degree bounded by D and bit size bounded by L, we can compute $\mathbf{SR}(A, B)(\mathsf{a})$, where $\mathcal{L}(\mathsf{a}) \leq L$, in $\widetilde{\mathcal{O}}_B(\mathsf{M}(D, L))$. In our case $D = \mathcal{O}(q)$ and $L = \mathcal{O}(p\tau + q\sigma)$, thus the total costs is $\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma)$.

It remains to compute the evaluation $\mathbf{SR}_{q+1}(\mathsf{a}) = f(\mathsf{a})$. This can be done using Horners' scheme in $\widetilde{\mathcal{O}}_B(p \max\{\tau, p\sigma\})$. Thus, the whole procedure has complexity

$$\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma + p\max\{\tau, p\sigma\}),$$

where the term $p\tau$ is dominated by $pq\tau$. □

PROOF OF COR. 2.5. Assume that α is not a common root of f and g in $[\mathsf{a}, \mathsf{b}]$, then it is known that

$$\operatorname{sign} g(\alpha) = [\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{a})) - \mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{b}))] \operatorname{sign}(f'(\alpha)).$$

Actually the previous relation holds in a more general context, when f *dominates* g, see [36] for details. Notice that $\operatorname{sign}(f'(\alpha)) = \operatorname{sign}(f(\mathsf{b})) - \operatorname{sign}(f(\mathsf{b}))$, which is known from the real root isolation process.

The complexity of the operation is dominated by the computation of $\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{a}))$ and $\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{b}))$, i.e. we compute $\mathbf{SQ}$ and evaluate it on a and b.

As explained above, there is no need to evaluate the polynomial of the biggest degree, i.e the first (and the second if $p < q$) of $\mathbf{SR}(f, g)$ over a and b. Thus the complexity is that of cor. 2.3, viz.

$$\widetilde{\mathcal{O}}_B(pq \max\{\tau_f, \tau_g\} + \min\{p, q\}^2 p\, \tau_f)$$

Thus the complexity of the operation is two times the complexity of the evaluation of the sequence over the endpoints of the isolating interval.

If α is a common root of f and g, or if f and g are not relative prime, then their gcd, which is the last non-zero polynomial in $\mathbf{SR}(f, g)$ is not a constant. Hence, we evaluate $\mathbf{SR}$ on a and b, we check if the last polynomial is not a constant and if it changes sign on a and b. If this is the case, then $\operatorname{sign}(g(\alpha)) = 0$. Otherwise we proceed as above. □

PROOF OF LEM. 2.6. Let there be $r \leq p$ real roots. The isolating point between two consecutive real roots α_j, α_{j+1} is of magnitude at most $\frac{1}{2}|\alpha_j - \alpha_{j+1}| := \frac{1}{2}\Delta_j$. Thus their product is $\frac{1}{2^r}\prod_j \Delta_j$. Using the Davenport-Mahler bound, $\prod_j \Delta_j \geq 2^{-\mathcal{O}(p^2 + p\tau_f)}$ and we take logarithms. □

PROOF OF LEM. 2.7. Let s_j be the bitsize of the j-th endpoint, where $0 \leq j \leq r$. The evaluation of $\mathbf{SR}(f, g)$ over this endpoint, by cor. 2.3, costs $\widetilde{\mathcal{O}}_B(pq\tau + \min\{p, q\}^2 s_j)$. To compute the overall cost, we should sum over all the isolating points. The first summand is $\widetilde{\mathcal{O}}_B(p^2 q\tau)$. By pr. 2.6, the second summand becomes $\widetilde{\mathcal{O}}_B(\min\{p, q\}^2(p^2 + p\tau_f))$ and is dominated. □

A.2 Multivariate polynomials

PROOF OF LEM. 2.9. Every polynomial in $\mathbf{SR}(f, g)$ has coefficients of magnitude bounded $2^{c(p+q)\tau}$, for a suitable constant c, assuming $\tau > \lg(d)$. Consider the map $\chi : \mathbb{Z}[y] \mapsto \mathbb{Z}$, such that $y \mapsto 2^{\lceil c(p+q)\tau \rceil}$, and let $\phi = \psi \circ \chi : \mathbb{Z}[y_1, y_2 \ldots, y_k] \to \mathbb{Z}$. Then $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leq \mathsf{c}(p + q)^k d\tau$. Now apply pr. 2.1. □

A.2.1 Real root counting

Let $F \in \mathbb{Z}[x, y]$, such that $\mathsf{dg}_x(F) = \mathsf{dg}_y(F) = n_1$ and $\mathcal{L}(F) = \sigma$. Let $\alpha, \beta \in \mathbb{R}_{alg}$, such that $\alpha = (A, [\mathsf{a}_1, \mathsf{a}_2])$ and $\beta = (B, [\mathsf{b}_1, \mathsf{b}_2])$, where $\mathsf{dg}(A), \mathsf{dg}(B) = n_2, \mathcal{L}(A), \mathcal{L}(B) \leq \tau$ and $\mathsf{c} \in \mathbb{Q}$, such that $\mathcal{L}(\mathsf{c}) = \lambda$. Moreover, assume that $n_1^2 = \mathcal{O}(n_2)$. We want to count the number of real roots of $\bar{F} = F(\alpha, y) \in (\mathbb{Z}(\alpha))[y]$ in $(-\infty, +\infty)$, in $(\mathsf{c}, +\infty)$ and in $(\beta, +\infty)$.

We may assume that the leading coefficient of $\bar{F}$ is nonzero. This is w.l.o.g. since we can easily check it, and/or we can use the good specialization properties of the subresultants [18, 13, 12].

Using Sturm's theorem, e.g. [3, 36], the number of real roots of $\bar{F}$ is $\mathtt{VAR}(\mathbf{SR}(\bar{F}, \bar{F}_y; -\infty)) - \mathtt{VAR}(\mathbf{SR}(\bar{F}, \bar{F}_y; +\infty))$. Hence, we have to compute the sequence $\mathbf{SR}(\bar{F}, \bar{F}_y)$ w.r.t. y, and evaluate it on $\pm\infty$, or equivalently to compute the signs of the principal subresultant coefficients, which lie in $\mathbb{Z}(\alpha)$.

The above procedure is equivalent, due to the good specialization properties of subresultants [3, 13], to that of computing the principal subresultant coefficients of $\mathbf{SR}(F, F_y)$, which are polynomials in $\mathbb{Z}[x]$, and to evaluate them over α. In other words the good specialization properties assure us that we can compute a nominal sequence by considering the bivariate polynomials, and then perform the substitution $x = \alpha$.

The sequence, $\mathbf{sr}$, of the principal subresultant coefficients can be computed in $\widetilde{\mathcal{O}}_B(n_1^4\sigma)$, using cor. 2.12 with $p = q = d = n_1$, and $\tau = \sigma$. The sequence $\mathbf{sr}$, contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x]$, each of degree $\mathcal{O}(n_1^2)$ and bitsize $\mathcal{O}(n_1\sigma)$. We compute the sign of each one evaluated over α in

$$\widetilde{\mathcal{O}}_B(n_1^2 n_2 \max\{\tau, n_1\sigma\} + n_2 \min\{n_1^2, n_2\}^2\tau)$$

using cor. 2.5 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$, and $\tau_g = n_1\sigma$. This proves the following:

LEMMA A.1. *We count the number of real roots of $\bar{F}$ in $\widetilde{\mathcal{O}}_B(n_1^4 n_2\sigma + n_1^5 n_2\tau)$.*

In order to compute the number of real roots of $\bar{F}$ in $(\beta, +\infty)$, we use again Sturm's theorem. The complexity of the computation is dominated by the cost of computing $\mathtt{VAR}(\mathbf{SR}(\bar{F}, \bar{F}_y; \beta))$, which is equivalent to computing $\mathbf{SR}(F, F_y)$ w.r.t. to y, which contains bivariate polynomials, and to compute their signs over (α, β). The cost of computing $\mathbf{SR}(F, F_y)$ is $\widetilde{\mathcal{O}}_B(n_1^5\sigma)$ using cor. 2.11 with $p = q = d = n_1$, and $\tau = \sigma$. The sequence contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x, y]$ of degrees $\mathcal{O}(n_1)$ and $\mathcal{O}(n_1^2)$, w.r.t. x and y respectively, and bitsize $\mathcal{O}(n_1\sigma)$. We can compute the sign of each of them evaluated it over (α, β) in

Algorithm 3: GRID(F, G)

Input: $F, G \in \mathbb{Z}[x, y]$
Output: The real solutions of $F = G = 0$

$R_x \leftarrow \mathtt{res}_y(F, G)$
$L_x, M_x \leftarrow$ SOLVE(R_x)
$R_y \leftarrow \mathtt{res}_x(F, G)$
$L_y, M_y \leftarrow$ SOLVE(R_y)
$Q \leftarrow \emptyset$
foreach $\alpha \in L_x$ **do**
  foreach $\beta \in L_y$ **do**
    if SIGN_AT$(F, \alpha, \beta) = 0 \wedge$ SIGN_AT$(G, \alpha, \beta) = 0$
    then $Q \leftarrow$ ADD$(Q, \{\alpha, \beta\})$
RETURN Q

$\widetilde{\mathcal{O}}_B(n_1^4 n_2^3 \max\{n_1\sigma, \tau\})$ (th. 2.14). This proves the following:

LEMMA A.2. *We can count the number of real roots of $\bar{F}$ in $(\beta, +\infty)$ in $\widetilde{\mathcal{O}}_B(n_1^5 n_2^3 \max\{n_1\sigma, \tau\})$.*

By a more involved analysis, taking into account the difference in the degrees of the bivariate polynomials, we can gain a factor. We omit it for reasons of simplicity.

Finally, in order to count the real roots of $\bar{F}$ in $(\mathsf{c}, +\infty)$, it suffices to evaluate the sequence $\mathbf{SR}(F, F_y)$ w.r.t. y on c, thus obtaining polynomials in $\mathbb{Z}[x]$ and compute the signs of these polynomials evaluated over α.

The cost of the evaluation $\mathbf{SR}(F, F_y; \mathsf{c})$ is $\widetilde{\mathcal{O}}_B(n_1^4 \max\{\sigma, \lambda\})$, using cor. 2.13 with $p = q = d = n_1$, $\tau = \sigma$ and $\sigma = \lambda$. The evaluated sequence contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x]$, of degree $\mathcal{O}(n_1^2)$ and bitsize $\mathcal{O}(n_1 \max\{\sigma, \lambda\})$. The sign of each one evaluated over α can be compute in

$$\widetilde{\mathcal{O}}_B(n_1^2 n_2 \max\{\tau, n_1\sigma, n_1\lambda\} + n_1^4 n_2 \tau),$$

using cor. 2.5 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$ and $\tau_g = n_1 \max\{\sigma, \lambda\}$. This leads to the following:

LEMMA A.3. *We can count the number of real roots of $\bar{F}$ in $(\mathsf{c}, +\infty)$ in $\widetilde{\mathcal{O}}_B(n_1^4 n_2 \max\{n_1\tau, \sigma, \lambda\})$.*

A.3 Bivariate real solving

PROOF OF LEM. 3.2. Suppose t_0 is such that the degree does not change. It suffices to find, among n^4 integer numbers, one that does not make Δ vanish; note that all candidate values are of bitsize $\mathcal{O}(\log n)$.

We perform the substitution $(x, y) \mapsto (x + ty, y)$ to F and G and we compute the resultant w.r.t. y in $\widetilde{\mathcal{O}}_B(n^5\sigma)$, which is a polynomial in $\mathbb{Z}[t, x]$, of degree $\mathcal{O}(n^2)$ and bitsize $\widetilde{\mathcal{O}}(d\sigma)$. We consider this polynomial as univariate in x and we compute first its square-free part, and then the discriminant of its square-free part. Both operations cost $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$ and the discriminant is a polynomial in $\mathbb{Z}[t]$ of degree $\mathcal{O}(n^4)$ and bitsize $\widetilde{\mathcal{O}}(d^4 + d^3\sigma)$.

We can evaluate the discriminant over all the first n^4 positive integers, in $\widetilde{\mathcal{O}}_B(n^8 + n^3\sigma)$, using the multipoint evaluation algorithm. Among these integers, there is at least one that is not a root of the discriminant. □

PROOF OF TH. 3.3. By the previous lemma, $t \in \mathbb{Z}$ is determined, with $\mathcal{L}(t) = \mathcal{O}(\log n)$, in $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$. Using this value, we isolate all the real roots of $R_t(x)$, defined in (1), and determine their multiplicities in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$. Let $\rho_j \simeq (R_t(x), [r_j, r'_j])$ be the real roots, for $j = 0, \ldots, r-1$.

By assumption, we have already isolated the roots of the system, denoted by $(\alpha_i, \beta_i) \in [a_i, a'_i] \times [b_i, b'_i]$, where $a_i, a'_i, b_i, b'_i \in \mathbb{Q}$ for $i = 0, \ldots, r-1$. It remains to match each pair (α_i, β_i) to a unique ρ_j by determining function $\phi : \{0, \ldots, r-1\} \to \{0, \ldots, r-1\}$, such that $\phi(i) = j$ iff $(\rho_j, \beta_i) \in \mathbb{R}^2_{alg}$ is a root of the sheared system and $\alpha_i = \rho_j + t\beta_i$.

Let $[c_i, c'_i] = [a_i, a'_i] - t[b_i, b'_i] \in \mathbb{Q}^2$. These intervals may be overlapping. Since the endpoints have bitsize $\mathcal{O}(n^4 + n^3\sigma)$, the intervals $[c_i, c'_i]$ are sorted in $\widetilde{\mathcal{O}}_B(n^6 + n^5\sigma)$. The same complexity bounds the operation of merging this interval list with the list of intervals $[r_j, r'_j]$. If there exist more than one $[c_i, c'_i]$ overlapping with some $[r_j, r'_j]$, some subdivision steps are required so that the intervals reach the bitsize of s_j, where 2^{s_j} bounds the separation distance associated to the j-th root. By pr. 2.6, $\sum_i s_i = \mathcal{O}(n^4 + n^3\sigma)$.

Our analysis resembles that of [10] for proving pr. 2.4. The total number of steps is $\mathcal{O}(\sum_i s_i) = \mathcal{O}(n^4 + n^3\sigma)$, each requiring an evaluation of $R(x)$ over a endpoint of size $\leq s_i$. This evaluation costs $\widetilde{\mathcal{O}}_B(n^4 s_i)$, leading to an overall cost of $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$ per level of the tree of subdivisions. Hence the overall complexity is bounded by $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma^2)$. □

A.4 Applications

PROOF OF COR. 4.2. Initially we compute the isolating interval representation of the real roots of $P = Q = 0$ in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$, using GRUR_SOLVE. There are $\mathcal{O}(n^2)$ real solutions, which are represented in isolating interval representation, with polynomials of degrees $\mathcal{O}(n^2)$ and bitsize $\mathcal{O}(n^2 + n\sigma)$.

For each real solution, say (α, β), for each polynomial A_i, B_j, C_k we compute the signs of $\mathrm{sign}(A_i(\alpha, \beta))$, $\mathrm{sign}(B_i(\alpha, \beta))$ and $\mathrm{sign}(C_i(\alpha, \beta))$. Each sign evaluation costs $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$, using th. 2.14 with $n_1 = n$, $n_2 = n^2$ and $\sigma = n^2 + n\sigma$. In the worst case we need n^2 of them, hence, the cost for all sign evaluations is $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell\, n^{11}\, \sigma)$. □

B. EXPERIMENTS

System R_1:

$$f = 1 + 2x - 2x^2y - 5xy + x^2 + 3x^2y$$
$$g = 2 + 6x - 6x^2y - 11xy + 4x^2 + 5x^3y$$

System R_2:

$$f = x^3 + 3x^2 + 3x - y^2 + 2y - 2$$
$$g = 2x + y - 3$$

System R_3:

$$f = x^3 - 3x^2 - 3xy + 6x + y^3 - 3y^2 + 6y - 5$$
$$g = x + y - 2$$

System M_1:

$$f = y^2 - x^2 + x^3$$
$$g = y^2 - x^3 + 2x^2 - x$$

System	SLV														
	GRID			MRUR							GRUR				
	Projections	Univariate	Bivariate	Projections	Univariate	StHa Sequence	Interm. Points	Filtering on x-axis	Compute K	FIND (Biv. Sol.)	Projections	Univariate	Interm. Points	Rational Bivariate	$\mathbb{R}_{alg}$ Bivariate
R_1	0.47	25.83	73.40	0.00	25.79	14.66	0.56	0.56	14.29	44.16	0.30	45.97	1.09	52.39	0.00
R_2	0.08	38.31	61.59	0.00	14.03	0.57	0.19	74.50	3.79	6.92	0.05	6.54	0.11	0.21	93.10
R_3	1.47	60.29	37.75	0.13	30.17	17.21	0.86	2.92	22.99	25.71	0.70	40.68	2.93	55.56	0.05
M_1	0.00	12.29	87.63	0.06	23.99	1.73	0.17	31.09	3.40	39.56	0.03	25.15	0.40	5.76	68.58
M_2	0.57	44.01	53.90	0.05	25.08	6.85	1.05	1.87	24.53	40.57	3.69	37.28	3.75	52.00	0.07
M_3	0.02	5.17	94.79	0.05	8.10	0.14	0.05	69.20	1.16	21.30	0.01	13.26	0.19	0.32	86.21
M_4	0.03	99.78	0.18	0.39	91.00	0.47	0.00	0.70	4.85	2.58	0.27	99.16	0.03	0.54	0.00
D_1	0.06	99.11	0.89	0.07	37.77	10.76	0.20	23.71	22.61	4.88	1.22	81.30	0.54	16.95	0.00
D_2	0.01	14.63	85.35	0.00	20.68	0.41	0.16	47.36	2.50	28.90	0.02	18.00	0.20	0.10	81.64
C_1	0.06	3.97	95.97	0.19	5.93	2.50	0.00	40.69	2.35	48.35	0.05	18.40	0.15	2.56	78.80
C_2	0.00	9.35	90.61	0.11	12.48	0.38	0.10	17.67	2.62	66.63	0.03	22.41	0.31	2.33	74.83
C_3	0.04	12.79	86.98	0.05	2.23	1.25	0.00	34.44	1.72	60.32	0.04	21.05	0.12	10.73	67.87
C_4	0.27	5.04	94.67	0.22	2.59	1.01	0.01	20.98	1.46	73.74	0.27	26.14	0.10	2.39	70.99
C_5	0.71	0.66	98.63	1.67	2.87	46.88	0.00	9.16	1.88	37.54	4.01	13.52	0.01	0.27	82.18
W_1	0.06	5.18	94.74	0.08	4.00	1.18	0.02	37.28	1.69	55.75	0.03	16.84	0.10	1.66	81.17
W_2	0.00	9.55	90.44	0.01	12.15	0.26	0.20	26.79	1.96	58.63	0.03	17.94	0.39	0.80	80.58
W_3	0.07	2.36	97.55	0.04	2.48	1.04	0.00	32.67	1.62	62.14	0.05	13.49	0.15	6.56	79.63
W_4	0.01	2.82	97.15	0.02	3.13	1.62	0.02	21.56	1.50	72.14	0.01	19.41	0.11	1.65	78.66
W_5	0.15	0.58	99.27	0.43	1.87	48.16	0.00	8.84	2.26	38.44	1.14	12.17	0.01	0.22	86.45

Table 3: Analyzing the percent of time required for various procedures in each algorithm. The table above presents the values in the *sheared case* (whenever it was necessary).

System M_2:

$$f = x^4 - 2x^2y + y^2 + y^4 - y^3$$
$$g = y - 2x^2$$

System M_3:

$$f = x^6 + 3x^4y^2 + 3x^2y^4 + y^6 - 4x^2y^2$$
$$g = y^2 - x^2 + x^3$$

System M_4:

$$f = x^9 - y^9 - 1$$
$$g = x^{10} + y^{10} - 1$$

System D_1:

$$f = x^4 - y^4 - 1$$
$$g = x^5 + y^5 - 1$$

System D_2:

$$f = -312960 - 2640x^2 - 4800xy - 2880y^2 + 58080x + 58560y$$
$$g = -584640 - 20880x^2 + 1740xy + 1740y + 274920x - 59160y$$

System C_1:

$$f = (x^3 + x - 1 - xy + 3y - 3y^2 + y^3)$$
$$(x^4 + 2x^2y^2 - 4x^2 - y^2 + y^4)$$
$$g = \text{diff}(f, y)$$

System C_2:

$$f = y^4 - 6y^2x + x^2 - 4x^2y^2 + 24x^3$$
$$g = \text{diff}(f, y)$$

System C_3:

$$f = ((x-1)^2 + y^2 - 2)((x+1)^2 + y^2 - 2)$$
$$((x-1)^2 + (y+2)^2 - 2)((x+1)^2 + (y+2)^2 - 2)$$
$$g = \text{diff}(f, y)$$

System C_4:

$$f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2)$$
$$(x^2 - 2x + 3 + y^2 + 4y)$$
$$(100000x^2 + 200000x + 299999 + 100000y^2 + 400000y)$$
$$g = \text{diff}(f, y)$$

System C_5:

$$f = (x^4 + 4x^3 + 6x^2 + 4x + y^4 + 4y^3 + 6y^2 + 4y)$$
$$(x^4 + 4x^3 + 6x^2 + 4x + y^4 - 4y^3 + 6y^2 - 4y)$$
$$(x^4 - 4x^3 + 6x^2 - 4x + y^4 + 4y^3 + 6y^2 + 4y)$$
$$(100000x^4 - 400000x^3 + 600000x^2 - 400000x$$
$$-1 + 100000y^4 - 400000y^3 + 600000y^2 - 400000y)$$
$$g = \text{diff}(f, y)$$

System W_1:

$$f = (x^3 + x - 1 - yx + 3y - 3y^2 + y^3)$$
$$(x^4 + 2y^2x^2 - 4x^2 - y^2 + y^4)$$
$$g = \text{diff}(f, y)$$

System W_2:

$$f = y^4 - 6y^2x + x^2 - 4x^2y^2 + 24x^3$$
$$g = \text{diff}(f, y)$$

System W_3:

$$f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2)$$
$$(x^2 - 2x + 3 + y^2 + 4y)(x^2 + 2x + 3 + y^2 + 4y)$$
$$g = \text{diff}(f, y)$$

System W_4:

$$f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2)$$
$$(x^2 - 2x + 3 + y^2 + 4y)$$
$$(100000x^2 + 200000x + 299999 + 100000y^2 + 400000y)$$
$$g = \text{diff}(f, y)$$

System W_5:

$$f = (x^4 + 4x^3 + 6x^2 + 4x + y^4 + 4y^3 + 6y^2 + 4y)$$
$$(x^4 + 4x^3 + 6x^2 + 4x + y^4 - 4y^3 + 6y^2 - 4y)$$
$$(x^4 - 4x^3 + 6x^2 - 4x + y^4 + 4y^3 + 6y^2 + 4y)$$
$$(100000x^4 - 400000x^3 + 600000x^2 - 400000x$$
$$-1 + 100000y^4 - 400000y^3 + 600000y^2 - 400000y)$$
$$g = \text{diff}(f, y)$$

C. SAMPLE USAGE

Our library requires a definition for variable `LIBPATH` which should point on the appropriate path where the source code is stored in your system. On the following, we assume that our library is located under `/opt/AlgebraicLibs/SLV/`. The following is an example for univariate solving:

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> read cat ( LIBPATH, "system.mpl" ):
> f := 3*x^3 - x^2 - 6*x + 2:
> sols := SLV:-solveUnivariate( f ):
> SLV:-display_1 ( sols );
  < x^2-2, [ -93/64, -45/32], -1.414213568 >
  < 3*x-1, [ 1/3, 1/3], 1/3 >
  < x^2-2, [ 45/32, 93/64], 1.414213568 >
```

Note, that the multiplicities of the roots do not appear, although they have been computed. Instead, the third argument of each component in the *printed* list is an approximation of the root. However, whenever possible we provide rational representation of the root.

The following is an example for bivariate solving, where the second root lies in $\mathbb{Z}^2$:

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> read cat ( LIBPATH, "system.mpl" ):
> f := 1+2*x+x^2*y-5*x*y+x^2:
> g := 2*x+y-3:
> bivsols := SLV:-solveGRID ( f, g ):
> SLV:-display_2 ( bivsols );
  < 2*x^2-12*x+1, [ 3, 7], 5.915475965 > ,
  < x^2+6*x-25, [ -2263/256, -35/4], -8.830718995 >

  < x-1, [ 1, 1], 1 > , < x-1, [ 1, 1], 1 >

  < 2*x^2-12*x+1, [ 3/64, 3/32], .8452400565e-1 > ,
  < x^2+6*x-25, [ 23179/8192, 2899/1024], 2.830943108 >
```

Again, just like in the case of univariate solving, the third argument that is printed on the component that describes each algebraic number is an approximation of the number and not the multiplicity of the root. Similarly, one could have used one of the other solvers on the above example by referring to their names, i.e. call the solvers with one of the following commands:

```
> bivsols := SLV:-solveMRUR ( f, g ):
> bivsols := SLV:-solveGRUR ( f, g ):
```

For those interested in the numerical values or rough approximations of the solutions one can get the appropriate output via `display_float_1` and `display_float_2` procedures. Hence, for the above examples we have:

```
> SLV:-display_float_1 ( sols );
  < -1.4142136 >
  <  0.3333333 >
  <  1.4142136 >
> SLV:-display_float_2 ( bivsols );
  [   5.9154759, -8.8309519, ]
  [   1.0000000,  1.0000000, ]
  [   0.0845241,  2.8309519, ]
```

Consider the list `sols` of $\mathbb{R}_{alg}$ numbers that was returned in the univariate case above; the following are examples on the usage of the `signAt` function provided by our Filtered Kernel[5]:

```
> FK:-signAt( 2*x + 3, sols[1] );
                                1
> FK:-signAt( x^2*y + 2, sols[3], sols[1] );
                               -1
```

Our class on Polynomial Remainder Sequences[6] exports functions allowing the computation of Subresultant and Sturm-Habicht sequences. Let $f, g \in \mathbb{Z}[x, y]$, then you can use *any* of the following commands in order to compute the desired PRS:

```
L := PRS:-StHa ( f, g, y ):
L := PRS:-StHaByDet ( f, g, y ):
L := PRS:-subresPRS ( f, g, y ):
L := PRS:-SubResByDet ( f, g, y ):
```

`PrintPRS` is used for viewing the PRS. For example, let f, g be those from the example on Bivariate Solving above:

```
> L := PRS:-subresPRS ( f, g, y ):
> PRS:-PrintPRS( L );
                 / 2      \                2
                 \x  - 5 x/ y + 1 + 2 x + x
                        y + 2 x - 3
                      3       2
                   2 x  - 14 x  + 13 x - 1
```

Finally, the variance of the above sequence evaluated at $(1, 0)$ can be computed by:

```
> G := PRS:-Eval ( L, 1, 0 );
                        G := [4, -1, 0]
> PRS:-var( G );
                               1
```

[5]Located in file: `FK.mpl`
[6]Located in file: `PRS.mpl`