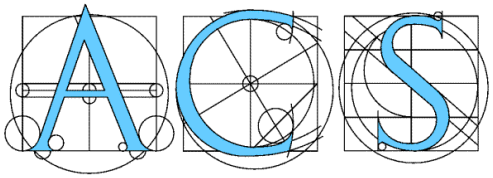

ACS

Algorithms for Complex Shapes
with Certified Numerics and Topology

Benchmarks and evaluation of experimental algebraic kernels

Dimitrios I. Diochnos

Ioannis Z. Emiris

Elias P. Tsigaridas

ACS Technical Report No.: ACS-TR-243306-02

Part of deliverable: WP-III/D3
Site: NUA
Month: 24

Project co-funded by the European Commission within FP6 (2002–2006)
under contract nr. IST-006413

Benchmarks and evaluation of experimental algebraic kernels

Dimitrios I. Diochnos* Ioannis Z. Emiris† Elias P. Tsigaridas‡§

April 17, 2007

Abstract

We present three projection based algorithms for real solving bivariate polynomial systems, as well as a detailed experimental analysis of their implementation in our MAPLE algebraic toolbox. Our implementation can lead to an algebraic kernel in CGAL because all necessary algebraic operations have been implemented. Finally, our experimental study includes comparative results with GBRS and three SYNAPS solvers (STURM, SUBDIV and NEWMAC).

1 Introduction

This report describes our MAPLE implementation and illustrates its capabilities through comparative experiments. Our library is open source software[1]. The design of the library is based on object oriented programming and the generic programming paradigm, common to `C++`, so as to be easy to transfer our implementation in `C++`, in the future.

The core object in our library is the class of real algebraic numbers, that are represented in isolating interval representations. We provide functionalities for computing signed polynomial remainder sequences with various algorithms; real solving univariate polynomials using Sturm's algorithm; computations with one and two real algebraic numbers, such as construction, sign evaluation, comparison; and, of course, our algorithms for real solving of bivariate polynomial systems. In order to speed up the computations we have implemented a filter layer, i.e, 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 [18]. We have not implemented, yet, the optimal algorithms for computing and evaluating polynomial remainder sequences.

*`d.diochnos@di.uoa.gr`

†`emiris@di.uoa.gr`

‡`et@di.uoa.gr`

§Currently at LORIA-INRIA Lorraine. Most part of this work was done while at NUA and INRIA Sophia-Antipolis.

[1] www.di.uoa.gr/~stud1098/SLV

2 Computing Sign of a bivariate polynomial.

Following the path from univariate solving we use Sturm Sequences in order to compute the sign of a polynomial evaluated at a planar Real Algebraic point. Given $\alpha \cong [A, [A_L, A_R]], \beta \cong [B, [B_L, B_R]]$ and $f \in \mathbb{Z}[x, y]$, we can compute the sign of $f(\alpha, \beta)$ as follows. We compute the Sturm Sequence of A and f wrt variable x. We create two duplicates of the above sequence and evaluate them at the rational endpoints $x = A_L$ and $x = A_R$ which define α. We then use Univariate SIGN_AT in order to compute the sign of each polynomial in the sequence at $y = \beta$. Finally, we compute the required sign by enumerating sign variations in the above polynomial sequences. Assuming $\deg_x(f) = \deg_y(f) = n_1$, $\deg(A) = \deg(B) = n_2$ and $\mathcal{L}(f) = \mathcal{L}(A) = \mathcal{L}(B) = \tau$, the complexity of the above algorithm is: $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \tau)$. The reader may refer to [7] for the details in the above algorithm and complexity analysis.

3 Bivariate real solving

Let $F, G \in \mathbb{Z}[x, y]$ be relatively prime and $\mathtt{dg}(F) = \mathtt{dg}(G) = n$ and $\mathcal{L}(F) = \mathcal{L}(G) = \sigma$. We present three algorithms and their complexity for real solving the system $F = G = 0$. The main idea behind the algorithms 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 the solutions. To project we use resultants and signed polynomial remainder sequences. The output of the algorithms is a list with pairs of real algebraic numbers and, if possible, the multiplicities of the solutions.

3.1 The GRID algorithm

Algorithm GRID, is straightforward and its first phase was also used in [19]. We compute the real algebraic numbers that correspond to 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 BIVARIATE-SIGN_AT by testing all rectangles in this grid.

The input of the algorithm is the polynomials $F, G \in \mathbb{Z}[x, y]$ and its 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.

The complexity of analysis of the algorithm can be found in [7]. The disadvantage of the algorithm is that exact implementation of its sub-algorithm BIVARIATE-SIGN_AT is not efficient. However, its simplicity makes it attractive and arithmetic filtering can speed up its implementation. The algorithm requires no genericity assumption on the input; however, it is possible to deterministically compute a shear transformation within the same complexity bound.

Last but not least, 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, which also reflects the sparseness of the equations. More importantly, we can count the real roots with a given abscissa α with application

of Sturm sequences.

Theorem 3.1 *Isolating all real roots of the system $F = G = 0$ using* GRID *has complexity $\widetilde{\mathcal{O}}_B(n^{14} + n^{13}\sigma + n^{10}\tau^2)$.*

We now examine the multiplicity of a root (α, β) of the system. Refer to [3, 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". More formally, the sum of multiplicities of all roots (α, β_j) equals the multiplicity of root $x = \alpha$ in the respective resultant polynomial. 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. Analysis of the deterministic shear computation can be found in [7] while previous work on the problem includes [9, 16, 20].

Lemma 3.2 *Computing shear transformation so that the resulting polynomial system is sufficiently generic, has complexity $\widetilde{\mathcal{O}}_B(n^9\sigma)$.*

After the application of the shear transformation we can compute the multiplicities of the roots of the sheared system. Then, we need to match the latter with the roots of the original system.

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

3.2 The M_RUR algorithm

M_RUR assumes that the polynomials are in generic position. Then:

Proposition 3.4 *[9, 2] Let F, G square-free and co-prime polynomials, in generic position. If $\mathbf{SR}_j(x, y) = \mathbf{sr}_j(x)y^j + \mathbf{sr}_{j,j-1}(x)y^{j-1} + \cdots + \mathbf{sr}_{j,0}(x)$, then if (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 $\beta = -\frac{1}{k}\frac{\mathbf{sr}_{k,k-1}(\alpha)}{\mathbf{sr}_k(\alpha)}$.*

Pr. 3.4 represents the ordinate of a solution in a Rational Univariate Representation (RUR) of the abscissa. The RUR applies to multivariate algebraic systems [4, 14, 5, 15, 6, 2] and generalizes the primitive element method [17], initiated by Kronecker. Here we adapt it to small-dimensional systems.

The algorithm that we present is similar to the algorithm of [10, 9]. However, notice their algorithm computes only a RUR of the roots using pr. 3.4, so the representation of the ordinates remains implicit. If we wish to compute with these numbers, this representation is not sufficient (we can always compute the minimal polynomial of the roots [11] in this algorithm, but this is highly inefficient). We modify the algorithm, so that the output includes isolating interval rectangles, hence the name modified-RUR (M_RUR).

Our most important difference with [9] is that they represent algebraic numbers by Thom's encoding while we use isolating intervals, which were thought of having high theoretical complexity. However, this is not the case [7].

We project on the x and the y-axis; for each real solution on the x-axis we compute its ordinate using pr. 3.4. First we compute the sequence $\mathbf{SR}(F,G)$ wrt y in $\widetilde{\mathcal{O}}_B(n^5\,\sigma)$. The projection on the two axis is similar to GRID algorithm. 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 roots in the x, resp. y axis. Moreover, we compute rational numbers q_j between the β_j's in $\widetilde{\mathcal{O}}_B(n^5\sigma)$:

$$q_0 < \beta_1 < q_1 < \beta_2 < \cdots < \beta_{\ell-1} < q_{\ell-1} < \beta_\ell < q_\ell, \tag{1}$$

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

The sub-algorithm COMPUTE_K: In order to apply pr. 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 [13, 9] and define recursively polynomials $\Gamma_j(x)$: Let $\Phi_0(x) = \frac{\mathbf{sr}_0(x)}{\mathsf{gcd}(\mathbf{sr}_0(x),\mathbf{sr}_0'(x))}$ and $\Phi_j(x) = \mathsf{gcd}(\Phi_{j-1}(x), \mathbf{sr}_j(x))$ 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 $k = j+1$.

The sub-algorithm FIND: Finally, we perform matching; the process takes a real root of R_x and computes the 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 (1): $q_j < A(\alpha) = \frac{A_1(\alpha)}{A_2(\alpha)} = \beta_j < q_{j+1}$. 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).

Theorem 3.5 *Let $F, G \in \mathbb{Z}[x,y]$ such that they are in generic position, their total degrees are bounded by n, and their bitsize by σ. If the polynomials are not relatively prime, the algorithm reports this and stops. Otherwise, it isolates all real roots of the system $F = G = 0$ with complexity $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$.*

The generic position assumption is without loss of generality since we can always put the system in such position by applying a shear transformation $(X,Y) \mapsto (X + tY, Y)$, where t is either a random number or computed deterministically, c.f [9, 16] and our remark on the previous section based on [7]. The bitsize of the (sheared) system becomes $\widetilde{\mathcal{O}}(n+\sigma)$ 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. We will report on this in a future work.

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 [18].

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 solution, 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.

Assuming fast multiplication algorithms, the cost of this operation is $\widetilde{\mathcal{O}}_B(n^4\sigma^2)$ and since we 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 $\widetilde{\mathcal{O}}_B(n + \sigma)$ [2].

Now for each α, we compute the polynomial $H = \gcd(\bar{F}, \bar{G})$. The cost of each operation is $\widetilde{\mathcal{O}}_B(n^6 + n^4\sigma^2)$ and the overall $\widetilde{\mathcal{O}}_B(n^8 + n^6\sigma^2)$. 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 intermediate point. Now, we consider this polynomial in $\mathbb{Z}[X]$ and we have to evaluate it over α. The cost of this operation is $\widetilde{\mathcal{O}}_B(n^6 + n^5\sigma + n^4 s_j)$. Summing over all the $\mathcal{O}(n^2)$ points, we obtain a complexity of $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$. Thus the overall complexity is $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$.

Again, the reader may refer to [7] for a detailed treatment on the complexity analysis.

Theorem 3.6 *We can isolate the real roots of the system $F = G = 0$, using* G_RUR *in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$.*

4 Augmenting Performance

The actual implementation uses some filtering techniques which are described in this section.

4.1 M_RUR pre-computation filtering

Since M_RUR will search for solutions along the y-axis we refine [1] the intervals of candidate solutions along the x-axis in order to help the interval arithmetic filters (refer to the following paragraph) that will be used inside the FIND procedure.

4.2 Interval Arithmetic

In cases where we want to compute the sign of a polynomial evaluated at a real algebraic number, we first try to yield result via Interval Arithmetic techniques. We apply this filter heuristically, based on total degree of the input polynomials several times, with a combination of quadratic refinement of the defining intervals [1] between executions in each loop.

4.3 GCD

In cases where the above filter fails to yield a result and we either want to compare two real algebraic numbers or perform univariate SIGN_AT we compute the gcd of the two polynomials that are invloved. By definition the gcd of the two polynomials has a root in (the intersection of) the intervals iff both polynomials have a same root, in which case the two numbers are equal, or the required sign is zero.

4.4 Concluding

If both of the above filtering techniques fail, we switch to exact and costly computations via Sturm sequences. However, in these computations we do not have to use the rational endpoints with higher bitsize that have arisen through the above filtering techniques; instead we use endpoints with smaller bitsize.

5 Evaluating Performance

In this section we will present the performance of the implemented algorithms as well as compare their running time with other algebraic packages. More specifically, we have tested GBRS, and three solvers from SYNAPS (STURM, SUBDIV and NEWMAC) in the polynomials found in section A of the appendix.

5.1 Comparative Performance.

Table 1 presents running times for the polynomial systems described in appendix. Polynomial systems R_i represent random polynomials of small dimension. In these cases we can observe that all solvers yield solutions in less than 40 msecs. SYNAPS solvers which are implemented in `C++` are faster in the average case, however SUBDIV and NEWMAC which are based on double precision arithmetic, are unstable and may provide incorrect solutions. Polynomial systems M_i and D_i have polynomials of total degree ranging between 2 and 10 and try to mimic polynomial systems that may arise in non-linear Computational Geometry problems, while at the same time we try to increase multiplicities and singularity points in the intersections or pump the coefficients of the input polynomials. Finally, polynomial systems C_i and W_i represent systems that have to be solved when computing the topology of a real plane algebraic curve. Polynomial systems R_i, M_i and D_i were drawn from [8], polynomial systems C_i were drawn from [10], and finally W_is were taken from [20].

system	deg		sheared	sols	Average Time (msecs)						
					SLV			GbRs	Synaps		
	f	g			grid	mrur	grur		sturm	subdiv	newmac
R_1	3	4	☑	2	10	11	6	27	6	1,372*	12
R_2	3	1		1	25	21	40	24	1	2	2
R_3	3	1		1	1	2	1	24	1	17	2*
M_1	3	3	☑	4	119	61	46	26	7	27*	2
M_2	4	2		3	6	6	5	25	1	317*	2
M_3	6	3	☑	5	2,627	850	441	32	950	12,660*	9
M_4	9	10		2	348	323	280	171	f.p.e.	5	384
D_1	4	5		1	7	11	6	30	4	fail	7
D_2	2	2		4	383	114	147	27	28	5	2
C_1	7	6	☑	6	2,259	918	257	91	1,198	63*	150*
C_2	4	3	☑	6	365	174	106	30	101	2,043*	6*
C_3	8	7	☑	13	309	1,723	148	61	154	111*	173*
C_4	8	7	☑	17	4,127	5,029	492	148	8,575	121*	343*
C_5	16	15	☑	17	353,512	48,327	6,130	6,106	>7 hrs	18,726*	6,392*
W_1	7	6	☑	9	2,813	1,945	404	96	2,368	106*	38
W_2	4	3	☑	5	843	263	218	30	160	1,235*	12*
W_3	8	7	☑	13	1,958	1,822	243	72	3,332	153*	272*
W_4	8	7	☑	17	8,912	4,988	747	162	26,779	169*	341*
W_5	16	15	☑	17	411,563	50,949	6,442	5,716	>7 hrs	19,238*	3,580*

Table 1: Running Times are averages over 10 runs.

Apart from the three solvers that were presented earlier, we also tested GBRS [15], which performs exact real solving using Gröbner basis and RUR, through its MAPLE interface, as well as 3 SYNAPS solvers: STURM is a very naive implementation of the GRID algorithm, based on a previous work of ours; SUBDIV implements the algorithm in [12], based on the Bernstein basis and `double` arithmetic; and NEWMAC, which is a general purpose solver based on normal-form algorithms and computations of eigenvectors using LAPACK, which computes all complex solutions. It should be noted however, that accurate timing in MAPLE is very 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 GBRS.

Our solvers GRID and M_RUR demonstrate a high fluctuation in runtimes, compared, e.g, to the stability of GBRS. On the one hand, GBRS is based on a fundamentally different approach to Real Solving making it exhibit these results. On the other hand, GBRS presents a similar (if not higher in relative measures) fluctuation on problems requiring more than 30 msecs if someone inspects running times on a ratio basis with a 30 msecs cut-off. These results are the outcome of the intrinsic complexity problems that come along with each polynomial system from our test-bed. In addition to that, similar observations on the performance can be made for all three SYNAPS solvers.

Regarding the performance of G_RUR, although on average it performs slower

than GBRS by a factor of around 3, it yields solutions in less than a second, apart from systems C_5 and W_5 where its performance is similar to that of GBRS. This discrepancy is expected since GBRS is not based entirely on high-level MAPLE coding, which is the case of our solvers.

Our three implemented algorithms have demonstrated the most robust behaviour, not only by replying within our specified time limits, but also no errors were generated during their execution. In the case of GBRS, some errors regarding the communication of the application with the MAPLE kernel were generated, especially on the difficult systems C_5 and W_5. STURM solver of SYNAPS, presented a floating point exception on the sheared system M_4 and failed to reply within our time limits in two more cases. It should be noted however, that STURM is a very naive implementation of the GRID algorithm, based on previous work of ours. Moreover, its inefficiency can be justified by the fact that it evaluates determinants in order to compute square-free parts. As for NEWMAC, some error is introduced since it is based on LAPACK for computing eigenvalues. This is also the case for the SUBDIV solver; we have to mention that this solver was originally made for computing self-intersections in the unit cube. In cases where NEWMAC or SUBDIV provided an incorrect solution set we indicate so in table 1 with an asterisk $*$.

5.1.1 Shear Transformation

Column `sheared` in table 1 indicates with a ☑ whether or not a shear transform was necessary so that the system is in Generic Position. In all cases where we had to make a shear trasformation, our first attempt in the deterministic computationwas successful; i.e. we performed the shear $(x, y) \mapsto (x + 3y, y)$. Computing times are shown in table 2. However, our experiments indicate that deterministic computation for parameter t in the shear trasformation $(x, y) \mapsto (x+ty, y)$ can be very expensive in practice even for polynomial systems of very low dimension. For example, the computation in system D_1, which is composed by polynomials of total degree no more than 5, takes 176 msecs, while all of our solvers yield a solution in less than 11 msecs. Hence, actual implementations should rely on a random choice for value t, which will suffice with probability 1. Note that running times in table 2 are the result of high-performance MAPLE built-in procedures, thereby inducing state-of-the-art practical performance.

5.2 The effect of filtering

As it can be observed from the description of our filters, we expect the implementations that rely more on univariate SIGN_AT as well as comparisons between real algebraic numbers to benefit more from the filters. Hence, filtering techniques provide on average a speedup of 10 in the GRID implementation. Moreover, M_RUR achieves on average a similar speedup factor with the additional pre-computation filtering technique. The precomputation filtering in M_RUR usually contributes by a factor of 2-3. Finally, the average speedup in the case of G_RUR is around 2, since this algorithm relies more on gcd computations rather than on sign determination.

system	total degree		$\deg_{\mathbf{t}}(\Delta)$	time (msecs)
	f	g		
R_1	3	4	18	32
R_2	3	1	6	4
R_3	3	1	0	4
M_1	3	3	16	8
M_2	4	2	4	8
M_3	6	3	110	44
M_4	9	10	5,402	1,402
D_1	4	5	180	176
D_2	2	2	12	8
C_1	7	6	364	1,252
C_2	4	3	28	16
C_3	8	7	174	572
C_4	8	7	308	5,576
C_5	16	15	7,620	2,095,007
W_1	7	6	332	840
W_2	4	3	42	20
W_3	8	7	190	296
W_4	8	7	328	812
W_5	16	15	7,724	1,018,864

Table 2: Time to determine deterministic shear t per test.

5.3 More on our implementation

In this section we will attempt a closer investigation on the performance and bottlenecks of our solvers. A percentage breakdown of the time required for each system in section A can be found in table 4. Hence, in GRID's case we have three columns. The first one `Projections` shows the percent of time needed to calculate the resultants w.r.t. variables x and y. The second column `Univariate` presents the percent of the overall computing time needed for univariate real solving on the computed resultants. Finally, the third column `Bivariate` presents the percent of overall computing time needed to perform bivariate SIGN_AT and hence decide whether or not a candidate solution is indeed a solution of the given polynomial system. In M_RUR's case we can find 7 columns. Again, the first two columns `Projections` and `Univariate` have the same meaning as in GRID's case. The third column `StHa Sequence` presents the percent of total computing time which was required in order to compute the Sturm-Habicht sequence of the given polynomials w.r.t. variable y. The column `Interm. Points` reflects the computation of the Intermediate Points on y-axis. Column `Filtering on` x`-axis` presents the percent required for our filtering technique on candidate solutions along the x-axis. Column `Compute K` presents the percent required to compute the appropriate index k for each candidate solu-

tion along the x-axis. Finally, column `FIND (Biv. Sol.)` presents the required percent for matching solutions via FIND procedure. In G_RUR's case we can find 5 columns. The first three `Projections`, `Univariate`, and `Interm. Points` represent the same values as in M_RUR's case. Column `Rational Bivariate` represents the time required to compute solutions of the given polynomial system when the abscissa or the ordinate are rational numbers. Finally, column $\mathbb{R}_{alg}$ `Bivariate` shows the percent required to compute solutions of the given system when neither the abscissa nor the ordinate is a rational number.

Moreover, table 3 provides a more conceptual view of table 4 describing crucial statistical properties.

	phase of the algorithm	range		median	mean	std dev
		min	max			
GRID	projections	00.00	01.47	00.06	00.21	00.36
	univ. solving	00.58	99.78	12.29	23.78	30.40
	biv. solving	00.18	99.27	90.61	75.86	30.56
	sorting	00.00	01.52	00.02	00.15	00.35
MRUR	projection	00.00	01.67	00.07	00.19	00.37
	univ. solving	01.87	91.00	12.48	17.18	20.48
	StHa seq.	00.14	48.16	01.62	08.27	14.35
	inter. points	00.00	01.05	00.10	00.19	00.30
	filter x-cand	00.56	74.50	26.79	26.42	20.85
	compute K	01.16	24.53	02.50	06.27	07.92
	biv. solving	02.58	73.74	44.16	41.49	21.57
GRUR	projections	00.01	04.01	00.05	00.63	01.16
	univ. solving	06.54	99.16	21.05	28.88	23.39
	inter. points	00.01	03.75	00.19	00.56	00.99
	rational biv.	00.10	55.56	02.39	11.21	18.71
	$\mathbb{R}_{alg}$ biv.	00.00	93.10	78.80	58.46	35.41
	sorting	00.00	03.21	00.09	00.26	00.70

Table 3: Statistics on the performance of the various phases for the our three projection-based algorithms; drawn from tab. 4

As far as GRID solver is concerned we can observe that 90% of the time required to yield a solution on input system is spent during the matching procedure and BIVARIATE-SIGN_AT procedure. There are cases however, where tables 4 and 3 imply that univariate solving of the resultants (projections) can be a bottleneck in the algorithm, but this is rather superfluous since univariate solving takes more than 25% of the computing time in cases where there are very little candidate solutions and hence the matching procedure will most likely succeed in every pair. Another indication of this, is the fact that in all systems but M_4, the total computing time did not exceed 25 msecs.

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 4: 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).

In the case of M_RUR we can observe that 70% of the computing time is spent in univariate solving on the projections as well as on the matching procedure via FIND. Computing the appropriate index k via COMPUTE_K typically takes less than 5% of the total computing time with descrepancies to this rule only on systems that take less than 11 msecs in total. Our pre-computation filtering technique takes on average 25% of the computing time, but as it has been stated it typically speeds up overall computing time by a factor of *at least* 2-3. It should also be noted that this technique currently aims to help more our filters as the dimension of the input polynomial increases. Hence, in low dimension we may observe very high percentages in this filtering step; e.g. system R_2, but in any case provides improvements in the overall computing time. Finally, as the total degree or the bitsize of the input polynomials increases, we can observe that the computation of the **StHa** sequence can take a substantial share of the computing time, reminding us that we have not implemented yet the optimal algorithms for this purpose; check for example systems C_5, W_5.

Similar is the case on the G_RUR. The most costly operation is that of bivariate matching with the gcd computation in the extension field, which typically takes around 80% of the total time. On the other hand, univariate solving on the projections occupies about 20% of the time. In cases where we observe a smaller fragment on the phase of bivariate solving we can observe that input polynomial systems, either have small total degree, or we have to check for matching among a small number of candidate solutions (possibly with high multiplicities). In a nutshell, once again we have indications that we need better implementations in the computation and evaluation of Sturm sequences, since the projection phase takes no more than 1% of the total time on average.

References

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

[2] S. Basu, R. Pollack, and M-F.Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2003.

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

[4] J. Canny. *The Complexity of Robot Motion Planning*. ACM – MIT Press Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1987.

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

[6] A. Dickenstein and I. Emiris, editors. *Solving Polynomial Equations: Foundations, Algorithms and Applications*, volume 14 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, May 2005.

[7] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, 2007.

[8] 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.

[9] 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.

[10] 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.

[11] R. Loos. Computing in algebraic extensions. In B. Buchberger, G. E. Collins, R. Loos, and R. Albrecht, editors, *Computer Algebra: Symbolic and Algebraic Computation*, pages 173–187. Springer-Verlag, 1983.

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

[13] 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.

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

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

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

[17] B. van der Waerden. *Modern Algebra*. Ungar, 1953. Vol. 1-2.

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

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

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

A Test Bed

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 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:

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

System C_2:

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

System C_3:

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

System C_4:

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

System C_5:

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

System W_1:

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

System W_2:

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

System W_3:

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

System W_4:

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

System W_5:

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