ACS

Algorithms for Complex Shapes
with Certified Numerics and Topology

Experimental implementation of more operations on algebraic numbers, possibly with the addition of numeric filters, and of robust operations on small polynomial systems

Dimitrios I. Diochnos

Ioannis Z. Emiris

Bernard Mourrain

Elias P. Tsigaridas

ACS Technical Report No.: ACS-TR-241405-02

Part of deliverable:	WP-I/D1
Sites:	NUA, INRIA
Month:	24

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

Experimental implementation of more operations on algebraic numbers, possibly with the addition of numeric filters, and of robust operations on small polynomial systems

Dimitrios I. Diochnos[*] Ioannis Z. Emiris[†] Bernard Mourrain[‡]
Elias P. Tsigaridas[§¶]

Abstract

We continue the work in [6] and present our MAPLE implementation of an algebraic toolbox capable of doing computations with one and two real algebraic numbers and real solving bivariate polynomial systems. In addition we describe new functions of the subpackage of the `C++` library SYNAPS for root isolation of univariate and multivariate polynomials. For this implementation we combine symbolic and numeric tools and illustrate their behavior on some classical family of polynomials.

1 Introduction

Our motivation comes from real solving polynomial systems of a small number of unknowns, usually 2 or 3. The problem of well-constrained algebraic system solving is fundamental. However, most of the algorithms treat the general case or consider solutions over an algebraically closed field [2, 5, 8, 11, 13, 14, 15, 16]. We focus on real solving in the bivariate case. This is important in several applications such as the computation of the Voronoi diagram of ellipses, the computation of the topology of plane and space real algebraic curves, algorithms involving complex shapes and non-linear computational geometry in general.

Our algorithms isolate all common real roots inside non-overlapping rational rectangles, and determine the intersection multiplicity per root. We proposed three projection-based algorithms and analyse their asymptotic bit complexity. This lead us to a bound of $\widetilde{\mathcal{O}}_B(N^{12})$, when ignoring polylogarithmic factors, whereas the previous record bound was $\widetilde{\mathcal{O}}_B(N^{16})$ [10], 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. Our main tool

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

[†]`emiris@di.uoa.gr`

[‡]`mourrain@sophia.inria.fr`

[§]`et@di.uoa.gr`

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

is signed subresultant sequences, extended to several variables by the technique of binary segmentation. We exploit the recent breakthroughs on univariate root isolation, which reduced complexity by at least one order of magnitude to $\widetilde{\mathcal{O}}_B(N^6)$ [3, 4, 7]. This brought complexity closer to $\widetilde{\mathcal{O}}_B(N^4)$, which is achieved by numerical methods, e.g. [18].

We have implemented in MAPLE a package for computations with real algebraic numbers and for implementing our algorithms for bivariate polynomial system real solving. The implementation 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.

The next section describes our main implementation decisions and Sec. 3 presents in detail the available functionality.

2 Implementation

This section describes the main implementation techniques that we use in our MAPLE implementation. 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, i.e we use a square-free polynomial and an isolating interval with rational endpoints. In addition we keep an multiprecision float approximation of real algebraic numbers, the use of which we describe in the sequel.

Besides, real algebraic numbers, of equal importance are the polynomial remainder sequences. We provide functionalities for computing signed polynomial remainder sequences with various algorithms, i.e Euclidean, primitive, monic, subresultant, reduced and Sturm-Habicht and moreover evaluation of a polynomial or a sequence of polynomials over rational numbers, counting sign variations in a sequence and various output capabilities. We also provide an algorithm based on Sturm sequences [7], for isolating the real roots of a univariate integer polynomial and thus constructing real algebraic numbers. We also plan to implement the algorithm based on continued fractions [20, 21].

We also provide algorithms for computation with one and two real algebraic numbers, i.e comparisons and sign evaluations; and, of course, our algorithms for real solving of bivariate polynomial systems. For GCD computations in an extension field we use the MAPLE package of van Hoeij and Monagan [22]. We have not implemented, yet, the optimal algorithms for computing and evaluating polynomial remainder sequences.

As for real solving bivariate polynomial systems, we have implemented the algorithms GRID, M_RUR and G_RUR, and experiments with these algorithms are presented in another report. In the next section we present in details, how our library can be used.

[1]currently available at `www.di.uoa.gr/~stud1098/SLV`

Besides the the various techniques of exact computation that we use in our library, one important feature is the use of filtering techniques, 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. To be more specific, an algebraic number is represented by a square-free polynomial an isolating interval and approximate value (using multiprecision floats, already available in MAPLE). When a computation is needed that involves an algebraic number, for example a sign evaluation, then we first try to compute the result of the computation using interval arithmetic and the approximate value that we have. If we do not manage to certify the answer, then we refine the approximation, and we try again. We repeat this procedure 3 times. If we still fail to provide a certified answer then we switch to an exact, however, slower method. Notice, that in all easy cases, i.e when the required result is far away from zero, we manage to compute the answer using only computations with intervals with multiprecision floats are endpoints. The speedup is significant, and a more detailed analysis concerning the gain in real solving bivariate polynomial systems is presented in another report.

3 Using the library

Our library requires a definition for variable `LIBPATH` which should point on the appropriate path where the source code is stored in our system. On the following, we assume that our library is located under `/opt/AlgebraicLibs/SLV/`. The following is an example for univariate solving. First we load the library, we construct a polynomial and finally we isolate its real roots. The main functionality for solving is under the MAPLE `module SLV`, and the way that we call the functions is similar to the use of namespaces in `C++`. The real solving function returns an ordered list of real algebraic numbers. When we print the real algebraic numbers, we output the square-free polynomial, the isolating interval and a multi-precision float approximation of the number. Moreover, whenever possible, we provide rational representation of the root.

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> 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 >
```

Our class on Polynomial Remainder Sequences[2] exports functions allowing the computation of signed polynomial remainder sequences. Let $f, g \in \mathbb{Z}[x, y]$, then you can use *any* of the following commands in order to compute the desired

[2]Located in file: `PRS.mpl`

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 ):
```

Notice that the sequences can be computed either using the algorithms based on pseudo-division or on determinant computations. Computation of signed polynomial remainder sequences using determinants are the only way of computing these sequences when the coefficient domain does not support exact division, i.e when the coefficients are intervals or multiprecision floats.

The function `PrintPRS` is used for viewing the PRS. For example

```
> f := 1+2*x+x^2*y-5*x*y+x^2:
> g := 2*x+y-3:
> 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
```

In order to evaluate the previous sequence at $(1,0)$ and count the sign variation we do the following:

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

The following is an example for bivariate solving, where the second root lies in $\mathbb{Z}^2$. Note that the output of the real solving function is a list of pairs of real algebraic numbers, sorted lexicographically.

```
> 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[3], i.e first we try to compute the sign using the approximations of the real algebraic numbers and interval arithmetic. If we can not then we switch to an exact method based on Sturm-Habicht sequences.

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

[3]Located in file: FK.mpl

4 Filtering techniques for root isolation

The filtering techniques that we described in the previous section and used in the MAPLE implementation are even more important when they applied to c++ code. In this section we will describe a number of filtering techniques that are used in SYNAPS for uunivariate and multivariate real solving. We will also briefly mention the theory behide the filtering techniques.

4.1 Univariate Subdivision Solvers

Our objective is to isolate the real roots of $f = \sum_{i=0}^{d} a_i x^i \in \mathbb{Q}[x]$., i.e. to compute intervals with rational endpoints that contain one and only one root of f, as well as the multiplicity of every real root.

Here is the general scheme of the subdivision solver that we consider, augmented appropriately so that it also outputs the multiplicities. It uses an external function $V(f, I)$, which bounds the number of roots of f in the interval I.

```
Real Root Isolation INPUT: A polynomial $f \in \mathbb{Z}[x]$, such that
$\mathsf{dg}(()f) = d$ and $\mathcal{L}(f) = \tau$.
OUTPUT: A list of intervals with rational endpoints, which contain one
and only one real root of $f$ and the multiplicity of every real root.

  1. Compute the square-free part of $f$, i.e. $f_{red}$
  2. Compute an interval $I_0 = (-B, B)$ with rational endpoints that contains
     all the real roots. Initialize a queue $Q$ with $I_0$.
  3. While $Q$ is not empty do
       a) Pop an interval $I$ from $Q$ and compute $v := V(f, I)$.
       b) If $v = 0$, discard $I$.
       c) If $v = 1$, output $I$.
       d) If $v \geq 2$, split $I$ into $I_L$ and $I_R$ and push them to $Q$.
  4. Determine the multiplicities of the real roots, using the square-free
     factorization of $f$.
```

In the Bernstein subdivision solver approach, we convert the polynomial f to a representation of degree d in Bernstein basis:

$$f(x) = \sum_i b_i B_i^d(x), \text{ and } B_i^d(x) = \left(\begin{array}{c} d \\ i \end{array} \right) x^i (1-x)^{d-i} \qquad (1)$$

where b_i is usually referred as controlling coefficients. Such conversion is done through a basis conversion [9]. The above formula can be generalized to an arbitrary interval $[a, b]$ by a variable substitution $x' = (b-a)x + a$. We denote by $B_d^i(x; a, b)\binom{d}{i}(x-a)^i(b-x)^{d-i}(b-a)^{-d}$ the corresponding Bernstein basis on $[a, b]$. There are several useful properties regarding Bernstein basis given besides the well-known *Convex-Hull Properties* and the *de Casteljau subdivision algorithm* [9]. One is related to Descartes rule of signs and involve the number of sign variation $V(\mathbf{b})$ of the sequence $\mathbf{b} = [b_1, \ldots, b_k]$ that we define recursively as follows:

$$V(\mathbf{b}_{k+1}) = V(\mathbf{b}_k) + \begin{cases} 1, \text{if } b_i b_{i+1} < 0 \\ 0, \text{else} \end{cases} \tag{2}$$

This yield a simple and yet efficient test for the existence of real roots in a given interval:

Proposition 4.1 *Given a polynomial $f(x) = \sum_i^n b_i B_i^d(x; a, b)$, the number N of real roots of f on $]a, b[$ is less than or equal to $V(\boldsymbol{b})$, where $\boldsymbol{b} = (b_i)_{i=0,\ldots,d}$ and $N \equiv V(\boldsymbol{b}) \mod 2$.*

With this proposition,

- if $V(\mathbf{b}) = 0$, the number of real roots of f in $[a, b]$ is 0;
- if $V(\mathbf{b}) = 1$, the number of real roots of f in $[a, b]$ is 1.

In order to analyze it, a partial inverse of Descartes' rule and lower bounds on the distance between roots of a polynomial have been used. It is proved that the complexity of isolating the roots of a polynomial of degree d, with integer coefficients of bit size $\leq \tau$ is bounded by $\mathcal{O}\left(d^4\tau^2\right)$ up to some poly-logarithmic factors. See [4, 7] for more details.

Notice that this localization algorithm extends naturally to B-splines, which are piecewise polynomial functions [9].

Another interesting property of the Bernstein basis, is the following:

Lemma 4.2 *Let $f = \sum_{i=0}^d b_i\, B_d^i(x; u, v)$, $g = \sum_{i=0}^d c_i\, B_d^i(x; u, v)$, and $b_i \leq c_i$ for $i = 0, \ldots, n$, then $f(x) \leq g(x)$ for all $x \in [u, v]$.*

This result, which is a simple consequence of the positiveness of $B_d^i(x; u, v)$ on $[u, v]$, has interesting practical consequences. If a polynomial is represented with large coefficients in the basis $(B_d^i(x; u, v))_{i=0,\ldots,d}$, one can enclose them into intervals involving less "complex" coefficients. The sign variation function can also be extended to polynomials with interval coefficients, by counting 1 sign variation for a sign sub-sequence $+, ?, -$ or $-, ?, +$; 2 sign variations for a sign sub-sequence $+, ?, +$ or $-, ?, -$; 1 sign variation for a sign sub-sequence $?, ?$, where $?$ is the sign of an interval containing 0. Again in this case, if a family $\overline{f}$ of polynomials is represented by the sequence of intervals $\bar{\mathbf{b}} = [\bar{b}_0, \ldots, \bar{b}_d]$ in the Bernstein basis of the interval $[u, v]$,

- if $V(\bar{\mathbf{b}}) = 1$, all the polynomials of the family $\overline{f}$ have one root in $[u, v]$,
- if $V(\bar{\mathbf{b}}) = 0$, all the polynomials of the family $\overline{f}$ have no roots in $[u, v]$.
- if $V(\bar{\mathbf{b}}) = K$, for any polynomial f of the family $\overline{f}$, we have $V(f, I) \leq K$.

We denote by $\omega(I) = |v - u|$ the width of the interval $I = [v, u]$ and $\omega(\bar{\mathbf{b}})$ the maximum of the widths of the coefficient intervals $\bar{\mathbf{b}}_i, i = 0, \ldots, d$.

This leads to the following algorithm: The interval representation can be

Real Root Approximation Using Interval Coefficients, with Fixed Precision Arithmetic. INPUT: An interval $I_0 = [u, v]$, a squarefree polynomial $f \in \mathbb{Q}[x]$, a precision $\epsilon > 0$, a threshold $\kappa > 0$.
OUTPUT: A list of intervals of size $< \epsilon$, which contain one and only one real root of f.

```
Initialize a queue $Q$ with $I_0$.
Convert the representation of $f$ in the Bernstein basis on $I_0$ into an interval representation $\bar{\mathbf{b}}$ with $\omega(\bar{\mathbf{b}}) < \kappa$.
While $Q$ is not empty do
    a) Pop an interval $I$ from $Q$ and compute $sv := V(\bar{\mathbf{b}}, I)$.
    b) If $sv = 0$, discard $I$.
    c) If $sv = 1$ and $\omega(I) < \epsilon$, output $I$.
    d) If $\omega(\bar{\mathbf{b}}) > \kappa$, convert $f$ into an interval representation $\bar{\mathbf{b}}$ on the Bernstein basis of $I$, with $\omega(\bar{\mathbf{b}}) < \kappa$.
    e) Else split $I$ into $I_L$ and $I_R$ and push them to $Q$.
```

based on `double` precision arithmetic (for instance rounding up and down to the nearest `double` number) in order to improve the performance of the solver. The threshold κ has to be adapted to this arithmetic. In this case, all the subdivision steps are performed with machine precision arithmetic and only in the conversion step, we use the extended (exact) arithmetic. This improve considerably the performance of the solver in practice, while still guarantying the result. This approach has been implemented in the library SYNAPS (see the class `SlvBzBdg` in the documentation).

4.2 Multivariate Bernstein Subdivision Solver

We consider now the problem of computing the solutions of a polynomial system

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_s(x_1, \ldots, x_n) = 0 \end{cases}$$

in a box $B := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$. The method for approximating the real roots of this system, that we describe now uses the representation of multivariate polynomials in Bernstein basis, analysis of sign variations and univariate solvers (Section 4.1). The output is a set of small-enough boxes, which contain these roots. This subdivision solver which can be seen as an improvement of the *Interval Projected Polyhedron* algorithm in [19], it is described in more details in [17]. Simple filtering techniques are described to improve the behavior of the algorithm.

In the following, we use the Bernstein basis representation of a multivariate polynomial f of the domain $I := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$:

$$f(x_1, \ldots, x_n) = \sum_{i_1=0}^{d_1} \cdots \sum_{i_n=0}^{d_n} b_{i_1,\ldots,i_n}\, B_{d_1}^{i_1}(x_1; a_1, b_1) \cdots B_{d_n}^{i_n} x(x_n; a_n, b_n).$$

Definition 4.3 *For any $f \in \mathbb{R}[\mathbf{x}]$ and $j = 1, \ldots, n$, let*

$$\begin{aligned} m_j(f; x_j) &= \sum_{i_j=0}^{d_j} \min_{\{0 \le i_k \le d_k, k \neq j\}} b_{i_1,\ldots,i_n}\, B_{d_j}^{i_j}(x_j; a_j, b_j) \\ M_j(f; x_j) &= \sum_{i_j=0}^{d_j} \max_{\{0 \le i_k \le d_k, k \neq j\}} b_{i_1,\ldots,i_n} x\, B_{d_j}^{i_j}(x_j; a_j, b_j). \end{aligned}$$

Theorem 4.4 (Projection Lemma) *For any $\mathbf{u} = (u_1, \ldots, u_n) \in I$, and any $j = 1, \ldots, n$, we have*

$$m(f; u_j) \le f(\mathbf{u}) \le M(f; u_j).$$

As a direct consequence, we obtain the following corollary:

Corollary 4.5 *For any root $\mathbf{u} = (u_1, \ldots, u_n)$ of the equation $f(\mathbf{x}) = 0$ in the domain I, we have $\underline{\mu}_j \le u_j \le \overline{\mu}_j$ where*

- *$\underline{\mu}_j$ (resp. $\overline{\mu}_j$) is either a root of $m_j(f; x_j) = 0$ or $M_j(f; x_j) = 0$ in $[a_j, b_j]$ or a_j (resp. b_j) if $m_j(f; x_j) = 0$ (resp. $M_j(f; x_j) = 0$) has no root on $[a_j, b_j]$,*
- *$m_j(f; u) \le 0 \le M_j(f; u)$ on $[\underline{\mu}_j, \overline{\mu}_j]$.*

The solver implementation contains the following main steps. It consists in

1. applying a preconditioning step to the equations;
2. reducing the domain;
3. if the reduction ratio is too small, to split the domain

until the size of the domain is smaller than a given epsilon.

The following important ingredients of the algorithm parametrize the implementation of the algorithm:

Preconditioner. It is a transformation of the initial system into a system, which has a better numerical behavior. Solving the system $\mathbf{f} = 0$ is equivalent to solving the system $M\,\mathbf{f} = 0$, where M is an $s \times s$ invertible matrix As such a transformation may increase the degree of some equations, with respect to some variables, it has a cost, which might not be negligible in some cases. Moreover, if for each polynomial of the system not all the variables are involved, that is if the system is sparse with respect to the variables, such a preconditioner may transform it into a system which is not sparse anymore. In this case, we would prefer a partial preconditioner on a subsets of the equations sharing a subset of variables. We consider Global transformations, which minimize the distance between the equations, considered as vectors in an affine space of polynomials of a given degree and Local straightening (for $s = n$), which transform locally the system $\mathbf{f}$ into a system $J^{-1}\mathbf{f}$, where $J = (\partial_{x_i} f_j(\mathbf{u}_0)_{1\leq i,j\leq s}$ is the Jacobian matrix of $\mathbf{f}$ at a point $\mathbf{u}$ of the domain I, where it is invertible.

It can be proved that the reduction based on the polynomial bounds m and M behaves like Newton iteration near a simple root, that is we have a quadratic convergence, with this transformation.

Reduction strategy, that is the technique used to reduce the initial domain, for searching the roots of the system. It can be based on Convex hull properties as in [19] or on Root localisation, which is a direct improvement of the convex hull reduction and consists in computing the first (resp. last) root of the polynomial $m_j(f_k; u_j)$, (resp. $M_j(f_k; u_j)$), in the interval $[a_j, b_j]$. The current implementation of this reduction steps allows us to consider the convex hull reduction, as one iteration step of this reduction process.

The guarantee that the computed intervals contain the roots of f, is obtained by controlling the rounding mode of the operations during the de Casteljau computation.

Subdivision strategy, that is technique used to subdivide the domain, in order to simplify the forthcoming steps, for searching the roots of the system. Some simple rules that can be used to subdivide a domain and reduce its size. The approach, that we are using in our implementation is the parameter domain bisection: The domain b is then split in half in a direction j for which $|b_j - a_j|$ is maximal. But instead of choosing the size of the interval as a criterion for the direction in which we split, we may choose other criterion depending also on the value the functions m_i, M_j or f_j (for instance where $M_j - m_j$ is maximal).

A bound for the complexity of this method is detailed in [17]. It involves metric quantities related to the system $\mathbf{f} = 0$, such as the Lipschitz constant of $\mathbf{f}$ in B, the entropy of its near-zero level sets, a bound d on the degree of the equations in each variable and the dimension n.

Filtering. Such approach can naturally be combined with enclosure based on interval representation, so that the subdivision steps are performed only with machine precision arithmetic. Such combination of symbolic and certified numeric techniques appear to very efficient in practice, and particularly use full in geometric problems. Moreover, that can naturally be extended to approximation implicit curves and surfaces [1, 12].

They have been implemented in the library SYNAPS (see for instance the class `SBDSLV` specifying the multivariate subdivision solver).

References

[1] L. Alberti, G. Comte, and B. Mourrain. Meshing implicit algebraic surfaces: the smooth case. In L. S. M. Maehlen, K. Morken, editor, *Mathematical Methods for Curves and Surfaces: Tromso'04*, pages 11–26. Nashboro, 2005.

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

[3] 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, School of Science, Beihang University, Beijing, China, 2005.

[4] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the descartes method. In *ISSAC '06: Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*, pages 71–78, New York, NY, USA, 2006. ACM Press.

[5] I. Emiris. A general solver based on sparse resultants, Mar. 1995. Available also as Tech. Report 3110, INRIA Sophia-Antipolis, Jan. 1997.

[6] I. Emiris and P. Tsigaridas. Robust operations on small polynomial systems. Technical Report ACS-TR-241405-01, NUA, 2006.

[7] 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, 2006. also available in www.inria.fr/rrrt/rr-5897.html.

[8] I. Z. Emiris and J. Verschelde. How to count efficiently all affine roots of a polynomial system. *Discrete Applied Math., Special Issue on Comput. Geom.*, 93(1):21–32, 1999.

[9] G. Farin. *Curves and surfaces for computer aided geometric design : a practical guide.* Comp. science and sci. computing. Acad. Press, 1990.

[10] 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.

[11] Y. Lakshman and D. Lazard. On the complexity of zero-dimensional algebraic systems. In T. Mora and C. Traverso, editors, *Effective Methods in*

Algebraic Geometry, volume 94 of *Progress in Mathematics*, pages 217–225, Boston, 1991. Birkhäuser. (Proc. MEGA '90, Livorno, Italy).

[12] C. Liang, B. Mourrain, and J. Pavone. Subdivision methods for 2d and 3d implicit curves. In *Computational Methods for Algebraic Spline Surfaces*. Springer-Verlag, 2006. To appear.

[13] B. Mourrain. Enumeration problems in geometry, robotics and vision. In L. Gonzalez-Vega and T. Recio, editors, *Effective Methods in Algebraic Geometry*, Progress in Mathematics. Birkhäuser, 1996. (Proc. MEGA '94, Santander, Spain).

[14] B. Mourrain. A new criterion for normal form algorithms. In M. Fossorier, H. Imai, S. Lin, and A. Poli, editors, *Proc. AAECC*, volume 1719 of *LNCS*, pages 430–443, 1999.

[15] B. Mourrain and V. Pan. Solving special polynomial systems by using structured matrices and algebraic residues. In F. Cucker and M. Shub, editors, *Proc. Workshop on Foundations of Computational Mathematics*, pages 287–304, Berlin, 1997. Springer-Verlag.

[16] B. Mourrain and V. Pan. Asymptotic acceleration of solving polynomial systems. In *Proc. ACM Symp. Theory of Computing*, pages 488–496. ACM Press, New York, 1998.

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

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

[19] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Comput. Aided Geom. Design*, 10(5):379–405, 1993.

[20] E. P. Tsigaridas and I. Z. Emiris. On the complexity of real root isolation using Continued Fractions. (submitted to TCS), Jun 2006.

[21] E. P. Tsigaridas and I. Z. Emiris. Univariate polynomial real root isolation: Continued fractions revisited. In Y. Azar and T. Erlebach, editors, *Proc. 14th European Symposium of Algorithms (ESA)*, volume 4168 of *LNCS*, pages 817–828, Zurich, Switzerland, 2006. Springer Verlag.

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