Robust algebraic methods for geometric computing

Angelos Mantzaflaris

▶ **To cite this version:**

Angelos Mantzaflaris. Robust algebraic methods for geometric computing. Symbolic Computation [cs.SC]. Université Nice Sophia Antipolis, 2011. English. NNT : . tel-00651672

HAL Id: tel-00651672
https://theses.hal.science/tel-00651672

Submitted on 14 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITY OF NICE - SOPHIA ANTIPOLIS

STIC DOCTORAL SCHOOL

INFORMATION AND COMMUNICATION SCIENCES & TECHNOLOGIES

P H D T H E S I S

to obtain the title of

Ph.D. of Science

of the University of Nice - Sophia Antipolis

Specialty : COMPUTER SCIENCE

Presented and defended by

Angelos MANTZAFLARIS

Robust algebraic methods for geometric computing

Prepared at INRIA Sophia-Antipolis, project GALAAD

Defended on October 3, 2011

Jury :

President of jury :	Adam PARUSINSKI	-	University of Nice, France
Thesis adviser :	Bernard MOURRAIN	-	INRIA Méditerranée, France
Examinator :	Ioannis EMIRIS	-	University of Athens, Greece
Reviewers :	Chandrajit BAJAJ	-	University of Texas at Austin, USA
	Bert JÜTTLER	-	Johannes Kepler University, Austria
	Mohab SAFEY EL DIN	-	U. of Pierre and Marie Curie, France

Robust algebraic methods for geometric computing

Abstract: Geometric computation in computer aided geometric design and solid modeling calls for solving non-linear polynomial systems in an approximate-yet-certified manner. We introduce new subdivision algorithms that tackle this fundamental problem. In particular, we generalize the univariate so-called continued fraction solver to general dimension. Fast bounding functions, unicity tests, projection and preconditioning are employed to speed up convergence. Apart from practical experiments, we provide theoretical bit complexity estimates, as well as bounds in the real RAM model, by means of real condition numbers.

A main bottleneck for any real solving method is singular isolated points. We employ local inverse systems and certified numerical computations to provide certification criteria to treat singular solutions. In doing so, we are able to check existence and uniqueness of singularities of a given multiplicity structure using verification methods, based on interval arithmetic and fixed point theorems.

Two major geometric applications are undertaken. First, the approximation of planar semi-algebraic sets, commonly occurring in constraint geometric solving. We present an efficient algorithm to identify connected components and, for a given precision, to compute polygonal and isotopic approximation of the exact set.

Second, we present an algebraic framework to compute generalized Voronoï diagrams, that is applicable to any diagram type in which the distance from a site can be expressed by a bi-variate polynomial function (anisotropic, power diagram etc). In cases where this is not possible (eg. Apollonius diagram, VD of ellipses and so on), we extend the theory to implicitly given distance functions.

Keywords: root isolation, continued fractions, singular isolated point, root deflation, arrangement of curves, semi-algebraic set, Voronoï diagram, subdivision algorithm

Méthodes algébriques robustes pour le calcul géométrique

Résumé: Le calcul géométrique en modélisation et en CAO nécessite la résolution approchée, et néanmoins certifiée, de systèmes polynomiaux. Nous introduisons de nouveaux algorithmes de sous-division afin de résoudre ce problème fondamental, calculant des développements en fractions continues des coordonnées des solutions. Au delà des exemples concrets, nous fournissons des estimations de la complexité en bits et des bornes dans le modèle de RAM réelle.

La difficulté principale de toute méthode de résolution consiste en les points singuliers isolés. Nous utilisons les systèmes locaux inverses et des calculs numériques certifiés afin d'obtenir un critère de certification pour traiter les solutions singulières. Ce faisant, nous sommes en mesure de vérifier l'existence et l'unicité des singularités d'une structure de multiplicité donnée.

Nous traitons deux principales applications géométriques. La première: l'approximation des ensembles semi-algébriques plans, apparaît fréquemment dans la résolution de contraintes géométriques. Nous présentons un algorithme efficace pour identifier les composants connexes et pour calculer des approximations polygonales et isotopiques à l'ensemble exact.

Dans un deuxième temps, nous présentons un cadre algébrique afin de calculer des diagrammes de Voronoi. Celui-ci sera applicable à tout type de diagramme dans lequel la distance à partir d'un site peut être exprimé par une fonction polynomiale à deux variables (anisotrope, diagramme de puissance etc). Si cela n'est pas possible (par exemple diagramme de Apollonius, VD des ellipses etc), nous étendons la théorie aux distances implicitement données.

Mots-clés: isolation des racines, fractions continues, point singulier isolé, déflation de racine, arrangement des courbes, ensemble semi-algébrique, diagramme de Voronoï , algorithme de sous-division

Preface

The present thesis is the outcome of studies conducted by the author from November, 2008 to September, 2011 at team GALAAD, INRIA Sophia-Antipolis, France, under the supervision of Dr. Bernard Mourrain.

Our motivation has been to combine modern algebraic tools with emerging techniques in computer aided geometric design in order, on one hand, to bring closer the fields of algebraic geometry and applied or algorithmic geometry and, on the other hand, to take a step towards advancing today's CAD/CAM industry. This exciting idea evolved in the previous European projects GAIA and GAIA II (2000–2005) and continued in 2008 with the launch of the SAGA project.

The text is developed in a way that chapters can be read individually, since each one contains extended preliminary details. However, the reader is invited to follow the natural flow of the text, in order to experience the bottom-up approach, that we also adopted in implementing these techniques.

The work done on polynomial system solving has resulted in articles in the field of symbolic and numeric computation, while the work on arrangements and Voronoï diagrams has resulted in articles in geometric modeling. Furthermore, our real solving techniques were implemented in the MATHEMAGIX C++ libraries, and new geometric algorithms were integrated into AXEL modeler. The writing of this thesis was funded by the European Community's *Seventh Framework Programme* [FP7/2007-2013], Marie Curie Initial Training Network SAGA (ShApes, Geometry and Algebra), grant agreement n^{o} [PITN-GA-2008-214584].

I would like to express my gratitude to my supervisor, Bernard Mourrain, who has provided tremendous insight and guidance on a variety of topics. The lessons learned by our daily interaction, by joining him to workshops and meetings, and by observing his style of work, sharpness in decisions, or the way he manages the GALAAD project-team are also indispensable.

I would like to thank the reviewers of this thesis for undertaking the reporting task, as well as for their timely work and useful comments. My best regards also go to all the members of the jury, for accepting to participate to it and for taking the trip to Sophia-Antipolis, in order to attend the defense.

Thanks also go to Ioannis Emiris, my master's adviser and mentor during this thesis, for all his support throughout these years. His courses on computer algebra and computational geometry were the seeds to my involvement in scientific research. I would also like to thank my good colleague and co-author Elias Tsigaridas for his support and help.

Also, thanks to all scientific partners of the SAGA consortium for their vision and optimism regarding the future of CAD/CAGD research in Europe and beyond.

In particular, I would like to thank my secondary hosts, that accepted my visits to their labs during my thesis; I. Emiris, B. Jüttler and T. W. Kim for respective visits of four, two and half a month. I have had a fruitful time with them and their team-members. I must mention Dimitris Diochnos, Vissarion Fisikopoulos, Tatjiana Kalinka, Christos Konaxis, Christos Sirgeloudis from Athens, Martin Aigner, Szilvia Béla, Carlotta Giannelli, Madalina Horodog, Mario Kapl, Stefan Kleiss, Elisa Pilgerstorfer, Martha Rossgatterer, Tino Schulz, Ulrike Schwarzmair, Birgit Strodthoff from Linz and Yeong-hwa Seo, Kittichai Suthunyatanakit, Rushan Ziatdinov from Seoul. Thank you all for making me feel at home.

Doing this thesis was the opportunity to meet several great individuals at GALAAD team and at INRIA in general. Greetings go to: Jerome Brachat and Lu Baa Thang for their kindness and daily company in the office. Laurent Busé (and Oversteam) for relaxing rock-evenings with his guitar at the brasserie, Evelyne Hubert for adventurous hikes on the mountains of Côte d'Azur, Alessandra Bernardi and Meriadeg Perrinel for establishing an afternoon-coffee culture in the team. Christina Bertone and Nicolas Botbol for refreshing afternoons by the beaches of Antibes. Also, Adrien Poteaux (now in Lille) for being a genuine friend, André Galligo for his friendship and that memorable afternoon of biking in Kyoto, Daouda N'Diatta for making the best bargain in the world for me when buying a car. Finally, Sophie Honnorat and Catherine Djerfi for the solid administrative support that they provided, as well as Magali Dijan for her kindness and gratitude.

Greetings also go to my colleagues from the CSSC team: George Markomanolis, Dimitris Simos and Zafeirakis Zafeirakopoulos. Thanks to Christos, Miltos, Markos and the other guys from the greek community in Nice for carefree Saturday-night drinks and Sunday-afternoon coffees. Also, to my good high-school friends from my hometown Filiatra, especially Thanasis and Thodoris, for all the nice moments that we shared during my escapes to Greece.

I would like to express my warmest feelings to my family, including on one hand my natural parents (Alexios and Anthoula), my two grandmothers (Anastasia's) and brothers (Vasilis and Tasos) and, on the other hand, my parents-in-law (Thodoris and Efi) and sister-in-law (Despoina). I equally give them all my gratitude for their love and support.

Undertaking this thesis has certainly been an exciting and greatly rewarding experience. The only disharmony was the fact that I had to live for around three years in distance from the love of my life, Asimina. Even though this has not always been easy for us, she has been a constant support to me throughout this time. The least I can do in return is to dedicate this thesis to her, and promise to never stay away from her again for so long time.

Angelos Mantzaflaris
Antibes, October 2011

Contents

CHAPTER 1

Introduction

Contents

Computer science emerged in the twentieth century, providing the ground for computational and efficiency aspects to evolve in mathematical theories. Algebra and geometry are apt examples of this fact. Indeed, applied algebra and geometry form today broad academic fields as well as the basis of a variety of applications in industry, through the use of computers.

Computer Aided Design (CAD) is widely applied to domains such as automobile, aeronautic, architecture, and so on. As a result, it has been a driving force for research in computational geometry, geometric computing and computer algebra. This thesis is positioned at the intersection of these fields, and combines rigorous algebraic tools, such as symbolic computation, duality theory and methods from real algebraic geometry, with fast hybrid techniques, eg. subdivision methods and approximate numerical computation. Our ultimate objective is to provide access to robust and efficient software for critical tasks in geometric computing, since this can lead to significant improvements in CAD applications.

After the present introductory chapter, there are two chapters dealing with polynomial system solving (Chapter 2) and singular point identification (Chapter 3). Then, two more chapters demonstrate the use of subdivision techniques, coupled with real solving, to the geometric problems of computation of arrangements of curves and semi-algebraic sets (Chapter 4), or general Voronoi diagrams (Chapter 5).

1.1 Problems and approaches

Geometric computation often relies on semi-algebraic set representations, involving polynomial equations and inequalities. A basic and important operation in this context is the construction of specific points on these geometric objects. Typically, points of intersection of two objects, singular points, points with tangents in

a specific direction, and so on, are needed in many algorithms to further process the geometric objects.

Classical algebraic tools for getting to these points include resultants and subdivision solvers. Resultants provide a compact way for characterizing the solvability of a polynomial system and lead to efficient ways for describing the common roots, or to perform elimination. Subdivision methods have a geometric flavor and are thus quite appealing to geometric computing, since they provide ways to localize real roots in a given bounded domain. They have seen important progress lately (cf. [11, 85]), and they have the advantage that they can provide fast approximations of the solutions of polynomial systems.

Algebraic geometry traditionally aims at exact solutions and hence dominantly uses the exact arithmetic paradigm. However, CAD-systems are based on double precision floating point numbers and allow the user to provide geometric tolerances defining when two points should be considered to be the same. In the presence of singularities, this task becomes cumbersome. In particular, typical numeric techniques that are integrated in today's commercial software, eg. Newton's method, need to be modified in non-trivial ways to achieve fast convergence, and subdivision solvers may be trapped into straying or looping, thus failing to validate a single singular point. Approximate inputs (eg. polynomials with floating point coefficients) are commonly encountered in CAD-systems, adding the extra problem of treating almost singular clusters of points. Consequently, in improving quality and performance of geometric algorithms, handling singular cases is a key issue.

Having efficient techniques to compute punctual solutions opens a range of possibilities for higher-dimension problems. Semi-algebraic sets define a wide class of geometric objects, including curves, surfaces and volumes. They occur naturally after performing Boolean operations between implicit models, or as the description of regions of validity for a physical problem, under polynomial constraints. A related problem is the counting and treatment of their connected components. Again, certain intersection, extremal or boundary points, defined as solutions of the involved polynomial equations, are the keys to deduce the *shape* of the object.

Frequently, 2D semi-agberaic sets are encountered in CAGD, as the result of appling elimination on parametric models, in which case one gets algebraic curves living in the parametric domain. These curves define curved regions that need to be approximated and used, eg. for trimming parametric surfaces. Gaps created by the trimming process may destroy continuity of the CAD model and eventually cause the manufacturing process to fail.

Computing with curved objects is less developed in computational geometry. Traditionally, linear primitives are the objects of study, and any curved shape has to be meshed into a large instance of such primitives, in order to be processed. When looking at Voronoï diagram computation, curved objects lead to bisectors that are given as algebraic curves, thus algebraic geometry machinery may be applied.

1.2 Outline and main results

In Chapter 2, we derive a new algorithm for real root isolation, that we call MCF, that generalizes the Continued Fraction (CF) algorithm of univariate polynomials. In doing so, we elaborate on a correspondence between the coefficients of a multivariate polynomial represented in the Bernstein basis and in a tensor-monomial basis, which leads to homography representations of polynomial functions, that use only integer arithmetic (in contrast to Bernstein basis) and are feasible over unbounded regions. Then, we study a method to split this representation and we obtain a subdivision scheme for the domain of multivariate polynomial functions. A partial extension of *Vincent's Theorem* for multivariate polynomials is presented, which allows us to prove the termination of the algorithm. Bounding functions, projection and preconditioning are employed to speed up the scheme. The resulting isolation boxes have optimized rational coordinates, corresponding to the first terms of the *continued fraction expansion* of the real roots. Finally, we present new complexity bounds for a simplified version of the algorithm in the bit complexity model, and also bounds in the real RAM model for a family of subdivision algorithms in terms of the real *condition number* of the system. Examples computed with our C++ implementation illustrate the practical aspects of the method.

The third chapter is devoted to singular solutions, i.e. solutions where the Jacobian determinant is rank-deficient. As mentioned previously, a single multiple point is typically counted as a collection of points, thus threatening the termination of the subdivision process. We develop a new symbolic-numeric algorithm for the certification of such singular isolated points, using their associated local ring structure and certified numerical computations. An improvement of a previous method to compute inverse systems is presented, which avoids redundant computation and reduces the size of the intermediate linear systems to solve. We derive a one-step deflation technique, from the description of the multiplicity structure in terms of differentials. The deflated system can be used in Newton-based iterative schemes with quadratic convergence. Starting from a polynomial system and a sufficiently small neighborhood, we obtain a criterion for the existence and uniqueness of a singular root of a given multiplicity structure, applying a well-chosen symbolic perturbation. Standard verification methods, based e.g. on interval arithmetic and a fixed point theorem, are employed to certify that there exists a unique perturbed system with an exact singular root in the domain. Applications to topological degree computation and to the analysis of real branches of an implicit curve illustrate the method.

Our first geometric application, regarding arrangements of curves and general semi-algebraic domains is presented in Chapter 4. In our work we present an algorithm to efficiently and in a *certified* way compute the connected components

of semi-algebraic sets given by intersection or union of conjunctions of bi-variate equalities and inequalities. For any given precision, this algorithm can also provide a polygonal and isotopic approximation of the exact set. The idea is to localize the *boundary curves* by subdividing the space and then deduce their shape within small enough cells using only boundary information. Then a systematic traversal of the boundary curve graph yields polygonal regions *isotopic* to the connected components of the semi-algebraic set. Space subdivision is supported by a kd-tree structure and localization is done using Bernstein representation. We conclude by demonstrating our C++ implementation in the computer algebra system MATHEMAGIX.

Chapter 5 deals with a new algorithm for the computation of general Voronoi Diagrams (VD's) constrained to a given domain. The method is applicable to any VD type in which the distance from a site can be expressed by a bi-variate polynomial function, notably the anisotropic VD or even VD's of complex sites. We use again the Bernstein form of polynomials and DeCasteljau's algorithm to subdivide the initial domain and isolate bisector domains or domains that may contain a Voronoi vertex. The efficiency of our algorithm is due to a filtering process based on bounding the distance functions over the subdivided domains. This allows to exclude functions (thus sites) that do not contribute locally to the lower envelope of the lifted diagram. After the filtering process the bisector curves are approximated by line segments, and vertices are computed by means of root isolation on the underlying polynomial system, giving overall certified polygonal description of each Voronoi cell.

We complete the presentation of the thesis with some concluding words, and an outlook to future developments.

CHAPTER 2

Mutivariate continued fraction solver

Contents

We propose a new adaptive algorithm for zero-dimensional polynomial system real solving that acts in monomial basis, and exploits the continued fraction expansion of (the coordinates of) the real roots. This yields optimal bit-size rational approximations of the real roots. All computations are performed with integers, thus this is a division-free algorithm. Also, we derive a partial generalization of Vincent's theorem to the multivariate case (Th. 2.9), and use it to prove termination of the algorithm. We perform a (bit) complexity analysis of the algorithm, when oracles for lower bounds and counting the real roots are available (Proposition 2.25).

In all cases the bounds that we derive for the multivariate case, match the best known ones for the univariate case, if we restrict ourselves to the univariate case. Finally, using an inclusion test based on α-theorems (Section 2.7), we provide an output-sensitive complexity bound in the arithmetic model, which involves the real condition number of the system.

This work is in the spirit of [80], and belongs to the more general family of subdivision algorithms. The novelty of our approach is that the proposed method computes with polynomials in the tensor-monomial basis algorithm, generalizes the univariate continued fraction algorithm, and does not assume generic position. Moreover, we apply a subdivision-based approach and exploit the sign properties of the Bernstein representation of the polynomials, by proving a correspondence between the latter and a specific sequence of homography transformations. A first version of this method appeared in [72], while an extended paper was published in [73].

2.1 State-of-the-art

The problem of computing roots of univariate polynomials is one of the most well studied problems in mathematics and computer science and has a long history, see [83] for an expository paper. The last years most of the efforts concentrated on subdivision-based algorithms, where the localization of the roots is based on simple tests such as *Descartes' Rule of Signs* and its variant in the Bernstein basis, e.g. [32, 41, 81]. There were a lot of developments on the Boolean complexity of the various approaches, in the case where the coefficients of the polynomials are integers, that allowed us to gain a good understanding of the behavior of the algorithms from a theoretical and a practical point of view. In addition, approximation and bounding techniques have been developed [8, 45, 86, 89] to improve the local speed of convergence to the roots.

Even more recently, new attention has been given to continued fraction algorithms (CF), see e.g. [95, 103, 104] and references therein. They differ from previous subdivision-based algorithms in that instead of bisecting a given initial interval and thus producing a binary expansion of the real roots, they compute the continued fraction expansions of them. The algorithm relies heavily on computations of lower bounds of the positive real roots, and different ways of computing such bounds lead to different variants of the algorithm. The best known worst-case complexity of CF is $\widetilde{\mathcal{O}}_B(d^4\tau^2)$ [76], while its average complexity is $\widetilde{\mathcal{O}}_B(d^3\tau)$ [104], thus being the only complexity result that matches, even in the average case, the complexity bounds of numerical algorithms [84]. Moreover, the algorithm seems to be the most efficient in practice [50, 104].

In the multivariate case the cardinality of the solution set may be infinite, i.e.

of positive dimension. Special methods exist to deal with this problem, see [91]. On the other hand, polynomial systems coming from applications that we focus usually describe a set of isolated points with real coordinates.

Another track of study, closely related to zero-dimensional solving is by elimination via multivariate resultants [18, 37, 40, 59], in particular using special formulae for structured systems [29, 39].

Subdivision methods for the approximation of isolated roots of multivariate systems have also been investigated but their analysis is much less advanced. In [96], the authors used tensor-product representation in Bernstein basis and domain reduction techniques based on the convex hull property to speed up the convergence and reduce the number of subdivisions. In [34], the emphasis is put on the subdivision process, and stopping criterion based on the normal cone attached to the surface patch. In [80, 85], this approach has been improved by introducing pre-conditioning and univariate-solver steps. The complexity of the method is also analyzed in terms of intrinsic differential invariants. Recently bounding techniques based on fat arcs and fat spheres have been used as a means of approximating algebraic curves and surfaces as well as for approximating real solutions in two or three dimensions [11]. For subdivision-based algorithms based on inclusion-exclusion tests we also refer the reader to [27, 113].

2.2 Notations and preliminaries

For a polynomial $f \in \mathbb{R}[x_1, .., x_n] = \mathbb{R}[\boldsymbol{x}]$, $\deg(f)$ denotes its total degree, while $\deg_{x_i}(f)$ denotes its degree with respect to x_i. Let $f(\boldsymbol{x}) = f(x_1, .., x_n) \in \mathbb{R}[x_1, .., x_n]$ with $\deg_{x_k} f = d_k$, $k = 1, .., n$. If not specified, we denote $d = d(f) = \max\{d_1, .., d_n\}$.

We are interested in isolating the real roots of a system of polynomials $f_1(\boldsymbol{x}), .., f_s(\boldsymbol{x}) \in \mathbb{Z}[x_1, .., x_n]$, in a box $I_0 = [u_1, v_1] \times \cdots \times [u_n, v_n] \subset \mathbb{R}^n$, $u_k, v_k \in \mathbb{Q}$. We denote by $\mathcal{Z}_{\mathbb{K}^n}(f) = \{p \in \mathbb{K}^n; f(p) = 0\}$ the solution set in $\mathbb{K}^n$ of the equation $f(x) = 0$, where the ground field $\mathbb{K}$ is $\mathbb{R}$ or $\mathbb{C}$.

For a homogeneous polynomial $f(x_0, \ldots, x_n) = \sum_{|\alpha|=d} c_\alpha \boldsymbol{x}^\alpha \in \mathbb{R}[x]$ of degree d, we define

$$\|f\| = \sqrt{\sum_{|\alpha|=d} |c_\alpha|^2 \binom{d}{\alpha}^{-1}}.$$

For an affine polynomial $f(x_1, \ldots, x_n)$ of degree d, we define $\|f\|$ as the norm of its homogenization in degree d. For a system $\boldsymbol{f} = (f_1, \ldots, f_n)$ of polynomials f_i of degree d_i, we define $\|f\| = (\|f_1\|^2 + \cdots + \|f_n\|^2)^{\frac{1}{2}}$. An important property of this norm is that it stays invariant under an unitary change of coordinates.

For $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{C}^n$, $\|v\| := (|v_1|^2 + \cdots + |v_n|^2)^{\frac{1}{2}}$, $\|v\|_\infty := \max\{|v_i|; i = 1, \ldots, n\}$.

For $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$, $x \in \mathbb{K}^n$ and $\rho > 0$, we denote by

- ◇ $B_{\mathbb{K}}(x, \rho) = \{y \in \mathbb{K}^n; \|y - x\| < \rho\}$ the ball of center x and radius ρ;
- ◇ $I_{\mathbb{K}}(x, \rho) = \{y \in \mathbb{K}^n; \|y - x\|_\infty < \rho\}$ the box of center x and radius ρ;

If $I = I_1 \times \cdots \times I_n \subset \mathbb{R}^n$, we denote by $I_{\mathbb{C}}$ the product $D_{\mathbb{C}}(I_1) \times \cdots \times D_{\mathbb{C}}(I_n) \subset \mathbb{C}^n$ of discs $D_{\mathbb{C}}(I_j) \subset \mathbb{C}$ of diameter I_j.

In what follows $\mathcal{O}_B$, resp. $\mathcal{O}$, means bit, resp. arithmetic, complexity and the $\widetilde{\mathcal{O}}_B$, resp. $\widetilde{\mathcal{O}}$, notation means that we are ignoring logarithmic factors. For $a \in \mathbb{Q}$, $\mathcal{L}(a) \geq 1$ is the maximum bit size of the numerator and the denominator. For a polynomial $f \in \mathbb{Z}[x_1, .., x_n]$, we denote by $\mathcal{L}(f)$ the maximum of the bit-size of its coefficients (including one bit for the sign). In the following, we will consider classes of polynomials such that $\log(d(f)) = \mathcal{O}(\mathcal{L}(f))$.

Also, to simplify the notation we introduce multi-indices, for the variable vector $\boldsymbol{x} = (x_1, .., x_n)$, $\boldsymbol{x^i} := x_1^{i_1} \cdots x_n^{i_n}$, the sum $\sum_{\boldsymbol{i}=\boldsymbol{0}}^{\boldsymbol{d}} := \sum_{i_1=0}^{d_1} \cdots \sum_{i_n=0}^{d_n}$, and $\binom{\boldsymbol{d}}{\boldsymbol{i}} := \binom{d_1}{i_1} \cdots \binom{d_n}{i_n}$. The tensor Bernstein basis polynomials of multidegree degree $\boldsymbol{d}$ of a box I are denoted $B(\boldsymbol{x}; \boldsymbol{i}, \boldsymbol{d}; I) := B_{d_1}^{i_1}(x_1; u_1, u_1) \cdots B_{d_n}^{i_n}(x_n; u_n, u_n)$ where $I = [\boldsymbol{u}, \boldsymbol{v}] := [u_1, v_1] \times \cdots \times [u_n, v_n]$.

We shall now describe the family of algorithms that we consider. The main ingredients are

- ◇ a suitable representation of the equations in a given (usually rectangular) domain, in terms of a basis of $\mathbb{Z}[\boldsymbol{x}]$, for instance a representation in the Bernstein basis or in the monomial basis;
- ◇ an algorithm to split the representation into smaller sub-domains;
- ◇ a reduction procedure to shrink the domain.

Different choices for each of these ingredients lead to algorithms with different practical behavior. The general process is summarized in Algorithm 1.

The instance of this general scheme that we are gong to analyze in this chapter, generalizes the continued fraction method for univariate polynomials; the realization of the main steps (b-e) can be summarized as follows:

b) Perform a precondition process and compute a lower bound on the roots of the system, in order to reduce the domain.

Algorithm 2.1: Subdivision scheme

Input: A set of equations $f_1, f_2, .., f_s \in \mathbb{Z}[\boldsymbol{x}]$ with only ponctual common solutions, represented over a domain I.

Output: A list of disjoint domains, each containing one and only one real root of $f_1 = \cdots = f_s = 0$.

Initialize a stack Q and add $(I, f_1, .., f_s)$ on top of it;

While Q is not empty do

a) Pop a system $(I, f_1, .., f_s)$ and:

b) Perform a precondition process and/or a reduction process to refine the domain.

c) Apply an exclusion test to identify if the domain contains no root.

d) Apply an inclusion test to identify if the domain contains a single root. In this case output $(I, f_1, .., f_s)$.

e) If both tests fail, then split the representation $(I, f_1, .., f_s)$ into a number of sub-domains and push them to Q.

---

c) Apply interval analysis or sign inspection to identify if some f_i has constant sign in the domain, i.e. if the domain contains no root.

d) Apply Miranda test to identify if the domain contains a single root. In this case output $(I, f_1, .., f_s)$.

e) If both tests fail, split the representation at $(1, .., 1)$ and continue.

In the following sections, we are going to describe more precisely these specific steps and analyze their complexity. In Section 2.3, we describe the representation of domains via homographies and the connection with the Bernstein basis representation. Subdivision, based on shifts of univariate polynomials, reduction and preconditioning are analyzed in Section 2.4. Exclusion and inclusion tests as well as a generalization of Vincent's theorem to multivariate polynomials, are presented in Section 2.5. In Section 2.6, we recall the main properties of Continued Fraction expansion of real numbers and use them to analyze the complexity of a subdivision algorithm following this generic scheme. In Section 2.7, we bound the complexity of the subdivision method using the α-inclusion test in terms of the real condition number of the system. We conclude the present chapter with examples produced by our C++ implementation in Section 2.8.

2.3 Representation: homographies

A widely used representation of a polynomial f over a rectangular domain is the tensor-Bernstein representation. De Casteljau's algorithm provides an efficient way to split this representation to smaller domains. A disadvantage is that, converting integer polynomials to Bernstein form results in rational or, if one uses machine numbers, approximate Bernstein coefficients. We follow an alternative approach that does not require basis conversion since it applies to monomial basis: We introduce a tensor-monomial representation, i.e. a representation in the monomial basis over th product of projective spaces $\mathbb{P}^1 \times \cdots \times \mathbb{P}^1$ and provide an algorithm to subdivide this representation analogously to the Bernstein case.

In a tensor-monomial representation a polynomial is represented as a tensor (higher dimensional matrix) of coefficients in the natural monomial basis, that is,

$$f(\boldsymbol{x}) = \sum_{i_1,..,i_n}^{d_1,..,d_n} c_{i_1..i_n} \boldsymbol{x}^{(i_1,..,i_n)} = \sum_{\boldsymbol{i}=\boldsymbol{0}}^{\boldsymbol{d}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}},$$

for every equation f of the system. Splitting this representation is done using homographies. The main operation in this computation is the Taylor shift.

Definition 2.1. *A homography (or Möbius transformation) is a bijective projective transformation $\mathcal{H} = (\mathcal{H}_1, .., \mathcal{H}_n)$, defined over $\mathbb{P}^1 \times \cdots \times \mathbb{P}^1$ as*

$$x_k \mapsto \mathcal{H}_k(x_k) = \frac{\alpha_k x_k + \beta_k}{\gamma_k x_k + \delta_k},$$

with $\alpha_k, \beta_k, \gamma_k, \delta_k \in \mathbb{Z}$, $\gamma_k \delta_k \neq 0$, $k = 1, .., n$.

Using simple calculations, we can see that the inverse

$$\mathcal{H}_k^{-1}(x_k) = \frac{-\delta_k x_k + \beta_k}{\gamma_k x_k - \alpha_k}$$

is also a homography, hence the set of homographies is a group under composition. Also, notice that if $\det \mathcal{H} > 0$ then, taking proper limits when needed, we can write

$$\mathbb{R}_+ \mapsto \mathcal{H}_k(\mathbb{R}_+) = \left[\frac{\beta_k}{\delta_k}, \frac{\alpha_k}{\gamma_k}\right] , \tag{2.1}$$

hence $H(f) : \mathbb{R}_+^n \to \mathbb{R}$,

$$H(f) := \prod_{k=1}^{n} (\gamma_k x_k + \delta_k)^{d_k} \cdot (f \circ \mathcal{H})(x)$$

is a polynomial defined over $\mathbb{R}_+^n$ that corresponds to the (possibly unbounded) box

$$I_H = \mathcal{H}(\mathbb{R}_+^n) = \left[\frac{\beta_1}{\delta_1}, \frac{\alpha_1}{\gamma_1}\right] \times \cdots \times \left[\frac{\beta_n}{\delta_n}, \frac{\alpha_n}{\gamma_n}\right], \tag{2.2}$$

of the initial system, in the sense that the zeros of the initial system in I_H are in one-to-one correspondence with the positive zeros of $H(f)$.

We focus on the computation of $H(f)$. We use the basic homographies $T_k^c(f) = f|_{x_k = x_k + c}$ (translation by c) or simply $T_k(f)$ if $c = 1$, $C_k^c(f) = f|_{x_k = cx_k}$ (contraction by c) and $R_k(f) = x_k^{d_k} f|_{x_k = 1/x_k}$ (reciprocal polynomial). These notations are naturally extended to variable vectors; for instance $T^{\mathbf{c}} = (T_1^{c_1}, .., T_n^{c_n})$, $c = (c_1, .., c_n) \in \mathbb{Z}^n$. Complexity results for these computations appear in the following sections. We can see that these three basic transformations suffice to compute any homography:

Lemma 2.2. *The group of homographies with coefficients in $\mathbb{Z}$ is generated by R_k, C_k^c, T_k^c, $k = 1, .., n$, $c \in \mathbb{Z}$.*

Proof. It can be verified that a $H_k(f)$ with arbitrary coefficients $\alpha_k, \beta_k, \gamma_k, \delta_k \in \mathbb{Z}$ is constructed as

$$H_k(f) = C_k^{\gamma_k} R_k C_k^{\delta_k} T_k R_k C_k^{\alpha_k/\gamma_k - \beta_k/\delta_k} T_k^{\beta_k/\delta_k}(f)$$

where the product denotes composition. We abbreviate $C_k^{1/c} = R_k C_k^c R_k$ and $T_k^{u/c} = C_k^c T_k^u C_k^{1/c}$, $u, c \in \mathbb{Z}$, e.g. $C_k^{1/c}(x_k) = \frac{x_k}{c}$ and $T_k^{u/c}(x_k) = x_k + \frac{u}{c}$.

□

Representation via homography is in an interesting correspondence to the Bernstein representation:

Lemma 2.3. *Let $f = \sum_{\boldsymbol{i}=0}^{\boldsymbol{d}} b_{\boldsymbol{i}} B_{\boldsymbol{i}}^{\boldsymbol{d}}(\boldsymbol{x}, I_H)$ the Bernstein expansion of f in the box I_H yielded by a homography H. If*

$$H(f) = C^{\gamma} R C^{\delta} T^1 R C^{\alpha/\gamma - \beta/\delta} T^{\beta/\delta}(f) = \sum_{\boldsymbol{i}=0}^{\boldsymbol{d}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}$$

then $c_{\boldsymbol{i}} = \binom{\boldsymbol{d}}{\boldsymbol{i}} \boldsymbol{\gamma}^{\boldsymbol{i}} \boldsymbol{\delta}^{\boldsymbol{d}-\boldsymbol{i}} b_i$.

Proof. Let $[u_k, v_k] = \left[\frac{\beta_k}{\delta_k}, \frac{\alpha_k}{\gamma_k}\right]$. For a tensor-Bernstein polynomial

$\binom{\boldsymbol{d}}{\boldsymbol{i}} \frac{1}{(\boldsymbol{v}-\boldsymbol{u})^{\boldsymbol{d}}(\boldsymbol{x}-\boldsymbol{u})^{P}\boldsymbol{i}}(\boldsymbol{v}-\boldsymbol{x})^{\boldsymbol{d}-\boldsymbol{i}}$ we compute

$$
\begin{aligned}
&C^{\boldsymbol{\gamma}} RC^{\boldsymbol{\delta}} T^{\mathbf{1}} RC^{\boldsymbol{v}-\boldsymbol{u}} T^{\boldsymbol{u}} \left(\binom{\boldsymbol{d}}{\boldsymbol{i}} \frac{1}{(\boldsymbol{v}-\boldsymbol{u})^{\boldsymbol{d}}} (\boldsymbol{x}-\boldsymbol{u})^{\boldsymbol{i}} (\boldsymbol{v}-\boldsymbol{x})^{\boldsymbol{d}-\boldsymbol{i}}\right) \\
&= C^{\boldsymbol{\gamma}} RC^{\boldsymbol{\delta}} RT^{\mathbf{1}} RC^{\boldsymbol{v}-\boldsymbol{u}} \left(\binom{\boldsymbol{d}}{\boldsymbol{i}} \frac{1}{(\boldsymbol{v}-\boldsymbol{u})^{\boldsymbol{d}}} x^{\boldsymbol{i}} (\boldsymbol{v}-\boldsymbol{u}-\boldsymbol{x})^{\boldsymbol{d}-\boldsymbol{i}}\right) \\
&= C^{\boldsymbol{\gamma}} RC^{\boldsymbol{\delta}} RT^{1} \left(\binom{\boldsymbol{d}}{\boldsymbol{i}} (\boldsymbol{x}-\mathbf{1})^{\boldsymbol{d}-\boldsymbol{i}}\right) \\
&= C^{\boldsymbol{\gamma}} RC^{\boldsymbol{\delta}} \left(\binom{\boldsymbol{d}}{\boldsymbol{i}} x^{\boldsymbol{i}}\right) = \binom{\boldsymbol{d}}{\boldsymbol{i}} \boldsymbol{\gamma}^{\boldsymbol{i}} \boldsymbol{\delta}^{\boldsymbol{d}-\boldsymbol{i}} x^{\boldsymbol{i}}
\end{aligned}
$$

as needed. □

Corollary 2.4. *The Bernstein expansion of f in I_H is*

$$\sum_{\boldsymbol{i}=0}^{\boldsymbol{d}} \frac{c_{\boldsymbol{i}}}{\binom{\boldsymbol{d}}{\boldsymbol{i}} \boldsymbol{\gamma}^{\boldsymbol{i}} \boldsymbol{\delta}^{\boldsymbol{d}-\boldsymbol{i}}} B(\boldsymbol{x}; \boldsymbol{i}, \boldsymbol{d}; I_H).$$

That is, the coefficients of $H(f)$ coincide with the Bernstein coefficients up to contraction and binomial factors.

Thus tensor-Bernstein coefficients and tensor-monomial coefficients in a subdomain of $\mathbb{R}^n$ differ only by multiplication by positive constants. In particular they are of the same sign. Hence this corollary allows us to take advantage of sign properties (e.g. the variation diminishing property) of the Bernstein basis that still hold in homography representation.

The resulting representation of the system consists of the transformed polynomials $H(f_1), .., H(f_n)$, represented as tensors of coefficients as well as $4n$ integers, $\alpha_k, \beta_k, \gamma_k, \delta_k$ for $k = 1, .., n$ from which we can recover the endpoints of the domain, using (2.2).

2.4 Subdivision and reduction

2.4.1 The subdivision step

We describe the subdivision step using the homography representation. This is done at a point $\boldsymbol{u} = (u_1, .., u_n) \in \mathbb{Z}^n_{\geq 0}$. It consists in computing up to 2^n new sub-domains (depending on the number of nonzero u_k's), each one having $\boldsymbol{u}$ as a vertex.

Given $H(f_1), .., H(f_s)$ that represent the initial system over some domain, we consider the partition of $\mathbb{R}^n_+$ defined by the hyperplanes $x_k = u_k$, $k = 1, .., n$. These

intersect at $\boldsymbol{u}$ hence we call this *partition at* $\boldsymbol{u}$. Subdividing at $\boldsymbol{u}$ is equivalent to subdividing the initial domain into boxes that share the common vertex $\mathcal{H}(\boldsymbol{u})$ and have faces either parallel or perpendicular to those of the initial domain.

We need to compute a homography representation for every domain in this partition. The computation is done coordinate wise; observe that for any domain in this partition we have, for all k, either $x_k \in [0, u_k]$ or $x_k \in [u_k, \infty]$. It suffices to apply a transformation that takes these domains to $\mathbb{R}_+$. In the former case, we apply $T_k^1 R_k C_k^{u_k}$ to the current polynomials and in the latter case we shift them by u_k, i.e. we apply $T_k^{u_k}$. The integers $\alpha_k, \beta_k, \gamma_k, \delta_k$ that keep track of the current domain can be easily updated to correspond to the new sub-domain.

We can make this process explicit in general dimension: every computed sub-domain corresponds to a binary number of length n, where the $k-$th bit is 1 if $T_k^1 R_k C^{u_k}$ is applied or 0 if $T_k^{u_k}$ is applied.

In our continued fraction algorithm the subdivision is performed at $\boldsymbol{u} = \mathbf{1}$.

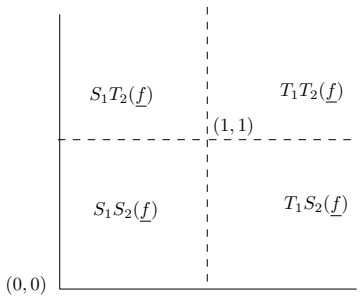

Figure 2.1: Subdividing the domain of $\boldsymbol{f}$.

Let us illustrate this process in dimension two. The system f_1, f_2 is defined over $\mathbb{R}^2_{>0}$. We subdivide this domain into $[0,1]^2$, $[0,1] \times \mathbb{R}_{>1}$, $\mathbb{R}_{>1} \times [0,1]$ and $\mathbb{R}_{>1} \times \mathbb{R}_{>1}$. Equivalently, we compute four new pairs of polynomials, as illustrated in Fig. 2.1 (we abbreviate $S_k = T_k^1 R_k$).

2.4.2 Complexity of subdivision step.

The transformation of a polynomial into two sub-domains, i.e. splitting with respect to one direction, consists in performing d^{n-1} univariate shifts, one for every coefficient $\in \mathbb{Z}[x_k]$ of $f \in \mathbb{Z}[x_k][x_1, .., \widehat{x_k}, .., x_n]$.

If the subdivision is performed in every direction, each transformation consists of d^{n-1} univariate shifts for every variable, i.e. nd^{n-1} shifts. There are 2^n sub-domains to compute, hence a total of $n^2 2^n d^{n-1}$ shifts have to be performed in a single subdivision step. We must also take into account that every time a univariate shift is performed, the coefficient bit-size increases.

The operations

$$T_k(f) = f|_{x_k=x_k+1} \text{ and } T_kR_k(f) = (x_k+1)^{d_k} f|_{x_k=\frac{1}{x_k+1}}$$

are essentially of the same complexity, except that the second requires one to exchange the coefficient of $c_{i_1,..,i_k,..,i_n}$ with $c_{i_1,..,d_k-i_k,..,i_n}$ before translation, i.e. an additional $\mathcal{O}(d^n)$ cost. Hence we only need to consider the case of shifts for the complexity.

The continued fraction algorithm subdivides a domain using unit shifts and inversion. Successive operations of this kind increase the bit-size equivalently to a big shift by the sum of these units. Thus it suffices to consider the general computation of $f(\boldsymbol{x}+\boldsymbol{u})$ to estimate the complexity of the subdivision step.

Lemma 2.5 (Shift complexity)**.** *The computation of $f(\boldsymbol{x}+\boldsymbol{u})$ with $\mathcal{L}(f)=\tau$ and $\mathcal{L}(u_k) \leq \sigma,\ k=1,..,n$ can be performed in $\widetilde{\mathcal{O}}_B(n^2d^n\tau + d^{n+1}n^3\sigma)$.*

Proof. We use known facts for the computation of $T_k^{u_k}(f)$ for univariate polynomials. If $\deg_k f = d_k$ and f is univariate, this operation is performed in $\widetilde{\mathcal{O}}_B(d_k^2\sigma + d_k\tau)$; the resulting coefficients are of bit-size $\tau + d_k\sigma$ [108]. Hence $f(x_1,..,x_k+u_k,..,x_n)$ is computed in $\widetilde{\mathcal{O}}_B(d^{n-1}(d_k^2\sigma + d_k\tau))$.

Suppose we have computed $f(x_1+u_1, x_{k-1}+u_{k-1}, x_k, .., x_n)$ for some k. The coefficients are of bit-size $\tau + \sum_{i=1}^{k-1}\sigma_i$. The computation of shift with respect to $k-$th variable $f(x_1+u_1,..,x_k+u_k,x_{k+1},..,x_n)$ results in a polynomial of bit-size $\tau+\sum_{i=1}^{k}\sigma_i$ and consists of $d^{n-1}\widetilde{\mathcal{O}}_B(d^2\sum_{i=1}^{k}\sigma_i + d\tau))$ operations. That is, we perform d^{n-1} univariate polynomial shifts, one for every coefficient of f in $\mathbb{Z}[x_k][x_1,..,\widehat{x_k},..,x_n]$.

This gives a total cost for computing $f(\boldsymbol{x}+\boldsymbol{u})$ of

$$d^{n-1}\sum_{k=1}^{n}\left(d^2\sum_{i=1}^{k}\sigma_i + d\tau\right) = nd^n\tau + d^{n+1}\sum_{k=1}^{n}(n+1-k)\sigma_k.$$

The latter sum implies that it is faster to apply the shifts with increasing order, starting with the smallest number u_k. Since $\sigma_k = \mathcal{O}(\sigma)$ for all k, and we must shift a system of $\mathcal{O}(n)$ polynomials we obtain the stated result. □

Remark An alternative way to compute a sub-domain using contraction, preferable when the bit-size of $\boldsymbol{u}$ is big is to compute $T_k^1C_k^u$ instead of T_k^u.

If we consider a contraction of factor u followed by a shift by 1 with respect to x_k for $\mathcal{O}(n)$ polynomials, we obtain $\widetilde{\mathcal{O}}_B(n^2d^n\tau + n^3d^{n+1} + nd^{n+1}\sigma)$ operations for the computation of the domain, where $\mathcal{L}(f)=\tau$, and $\mathcal{L}(u_k)\leq\sigma,\ k=1,..,n$.

The disadvantage is that the resulting coefficients are of bit-size $\mathcal{O}(\tau+d\sigma)$ instead of $\mathcal{O}(\tau+n\sigma)$ with the use of shifts. Also observe that this operation would

compute an expansion of the real root which differs from continued fraction expansion. As a consequence, using this transformation, a complexity analysis based on the properties of continued fractions, like the one we present in 2.6 would not apply.

Nevertheless in practice, this method has sometimes the effect of separating more easily the roots and thus lead to faster computation of isolation sub-domains. We have experimented with this solution, as one variant of our implementation.

2.4.3 Reduction: Bounds on the range of f

In this section we define univariate polynomials whose graph in $\mathbb{R}^{n+1}$ bounds the graph of f. For every direction k, we provide two polynomials bounding the values of f in $\mathbb{R}^n$ from below and above respectively.

Define

$$m_k(f; x_k) = \sum_{i_k=0}^{d_k} \min_{i_1,..,\widehat{i_k},..,i_n} c_{i_1..i_n} x_k^{i_k} \tag{2.3}$$

$$M_k(f; x_k) = \sum_{i_k=0}^{d_k} \max_{i_1,..,\widehat{i_k},..,i_n} c_{i_1..i_n} x_k^{i_k} \tag{2.4}$$

Lemma 2.6. *For any $x \in \mathbb{R}^n_+$, $n > 1$ and any $k = 1, .., n$, we have*

$$m_k(f; x_k) \leq \frac{f(x)}{\prod_{s\neq k}^{d_s} \sum_{i_s=0} x_s^{i_s}} \leq M_k(f; x_k). \tag{2.5}$$

Proof. For $x \in \mathbb{R}^n_+$, we can directly write

$$f(x) \leq \left(\sum_{i_k=0}^{d_k} \max_{i_1,..,\widehat{i_k},..,i_n} c_{i_1..i_n} x_k^{i_k} \right) \prod_{s\neq k}^{d_s} \sum_{i_s=0} x_s^{i_s}$$

The product of power sums is greater than 1; divide both sides by it. Analogously for $M_k(f; x_k)$. □

Corollary 2.7. *Given $k \in \{1, .., n\}$, if $u \in \mathbb{R}^n_+$ with $u_k \in]0, \mu_k]$, where*

$$\mu_k = \begin{cases} \textit{min. pos. root of } M_k(f, x_k) & \textit{if } M_k(f; 0) < 0 \\ \textit{min. pos. root of } m_k(f, x_k) & \textit{if } m_k(f; 0) > 0 \\ 0 & \textit{otherwise} \end{cases},$$

then $f(u) \neq 0$. Consequently, all positive roots of f lie in $\mathbb{R}_{>\mu_1} \times \cdots \times \mathbb{R}_{>\mu_n}$. Also, for $u \in \mathbb{R}^n_+$ with $u_k \in [\mathcal{M}_k, \infty]$,

$$\mathcal{M}_k = \begin{cases} \textit{max. pos. root of } M_k(f, x_k) & \textit{if } M_k(f;\infty) < 0 \\ \textit{max. pos. root of } m_k(f, x_k) & \textit{if } m_k(f;\infty) > 0 \\ \infty & \textit{otherwise} \end{cases},$$

it is $f(u) \neq 0$, i.e. all positive roots are in $\mathbb{R}_{<\mathcal{M}_1} \times \cdots \times \mathbb{R}_{<\mathcal{M}_n}$.

Combining both bounds we deduce that $[\mu_1, \mathcal{M}_1] \times \cdots \times [\mu_n, \mathcal{M}_n]$ is a bounding box for $f^{-1}(\{0\}) \cap \mathbb{R}^n_+$.

Proof. The denominator in (2.5) is always positive in $\mathbb{R}^n_+$. Let $\boldsymbol{u} \in \mathbb{R}^n$ with $u_k \in [0, \mu_k]$. If $M_k(f, 0) < 0$ then also $M_k(f, u) < 0$ and it follows $f(\boldsymbol{u}) < 0$. Similarly $m_k(f, 0) > 0 \Rightarrow m_k(f, u) > 0 \Rightarrow f(\boldsymbol{u}) > 0$. The same arguments hold for $[\mathcal{M}_k, \infty]$, $M_k(f;\infty) = R(M_k(f; x_k))(0)$, $m_k(f;\infty) = R(m_k(f; x_k))(0)$, and $R(f)$, since lower bounds on the zeros of $R(f)$ yield upper bounds on the zeros of f. □

Thus lower and upper bounds on the $k-$th coordinates of the roots of $(f_1, .., f_s)$ are given by

$$\max_{i=1,..,s} \{\mu_k(f_i)\} \quad \text{and} \quad \min_{i=1,..,s} \{\mathcal{M}_k(f_i)\} \tag{2.6}$$

respectively, i.e. the intersection of these bounding boxes.

We would like to remain in the ring of integers all along the process, thus integer lower or upper bounds will be used. These can be the floor or ceiling of the above roots of univariate polynomials, or even known bounds for them, e.g. Cauchy's bound.

If the minimum and maximum are taken with the ordering of coefficients defined as $c_i \prec c_j \iff c_i \binom{d}{j} \gamma^j \delta^{d-j} < c_j \binom{d}{i} \gamma^i \delta^{d-i}$ then different $m_k(f, x_k), M_k(f, x_k)$ polynomials are obtained. By Corollary 2.4 their control polygon is the lower and upper hull respectively of the projections of the tensor-Bernstein coefficients to the $k-$th direction and are known to converge quadratically to simple roots when preconditioning (described in the following paragraph) is utilized [80, Corollary 5.3].

Complexity analysis. The analysis of the subdivision step in Section 2.4.3 applies as well to the reduction step, since reducing the domain means to compute a new sub-domain and ignore the remaining volume of the initial box.

If a lower bound $\boldsymbol{l} = (l_1, \ldots, l_n)$ is known, with $\mathcal{L}(l_k) = \widetilde{\mathcal{O}}(\sigma)$, then the reduction step is performed in $\widetilde{\mathcal{O}}_B(n^2 d^n \tau + d^{n+1} n^3 \sigma)$. This is an instance of Lemma 2.5.

The projections of Lemma 2.6 are computed using $\mathcal{O}(d^n)$ comparisons. The computation of $\boldsymbol{l}$ costs $\widetilde{\mathcal{O}}_B(d^3\tau)$ in average, for solving these projections using

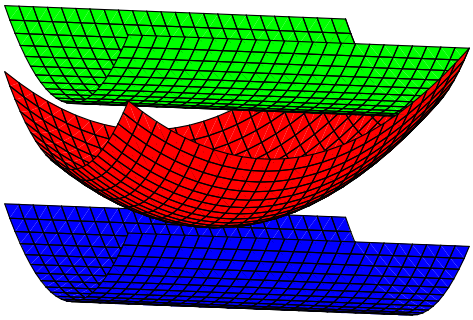

Figure 2.2: The enveloping polynomials $M_1(x)$, $m_1(x)$ in domain I_0 for a bi-quadratic polynomial $f(x, y)$.

univariate CF algorithm. Another option would be to compute well known lower bounds on their roots, for instance Cauchy's bound in $\mathcal{O}(d)$.

Illustration. Consider a bi-quadratic $f_0 \in \mathbb{R}[x, y]$, namely, $\mathsf{deg}_{x_1} f_0 = \mathsf{deg}_{x_2} f_0 = 2$ with coefficients c_{ij}. Suppose that $f_0 = H(f)$ for $I_0 = I_H$. We compute

$$m_1(f, x_1) = \sum_{i=0}^{2} \min_{j=0,\ldots,2} c_{ij}\, x_1^i \quad \text{and} \quad M_1(f, x_1) = \sum_{i=0}^{2} \max_{j=0,\ldots,2} c_{ij}\, x_1^i.$$

thus $m(f, x_1) \leq \frac{f(x_1,x_2)}{1+x_2+x_2^2} \leq M_1(f, x_1)$. Fig. 2.2 shows how these univariate quadratics bound the graph of f in I_0.

2.4.4 Preconditioning

To improve the reduction step, we use preconditioning. The aim of a preconditioner is to tune the system so that it can be tackled more efficiently; in our case we aim at improving the bounds of Corollary 2.7.

A preconditioning matrix P is an invertible $s \times s$ matrix that transforms a system $(f_1, .., f_s)^t$ into the equivalent one $P \cdot (f_1, .., f_s)^t$. This transformation does not alter the roots of the system, since the computed equations generate the same ideal. The bounds obtained on the resulting system can be used directly to reduce the domain of the equations before preconditioning. Preconditioning can be performed to a subset of the equations.

Since we use a reduction process using Corollary 2.7 we want to have among our equations n of them whose zero locus $f^{-1}(\{0\})$ is orthogonal to the $k-$th direction, for all k.

Assuming a square system, we precondition $H(f_1), .., H(f_n)$ to obtain a locally orthogonal to the axis system; an ideal preconditioner would be the Jacobian of the

system evaluated at a common root; instead, we evaluate $J_{H(f)}$ at the image of the center $\boldsymbol{u}$ of the initial domain I_H, $u_k = \frac{\alpha_k\delta_k+\beta_k\gamma_k}{2\gamma_k\delta_k}$. Thus we must compute the inverse of the Jacobian matrix $J_{H(\boldsymbol{f})}(x) = [\partial_{x_i} H(f_j)(x)]_{1\leq i,j,\leq n}$ evaluated at $\boldsymbol{u}' := \mathcal{H}(\boldsymbol{u}) = (\delta_1/\gamma_1, .., \delta_n/\gamma_n)$.

Precondition step complexity. Computing $J_{H(f)}(\boldsymbol{u}) \cdot (H(f_1), .., H(f_n))^t$ is done with cost $\widetilde{\mathcal{O}}_B(n^2d^n)$ and evaluating at u' has cost $\widetilde{\mathcal{O}}_B(n^2d^{n-1})$. We also need $\widetilde{\mathcal{O}}_B(n^2)$ for inversion and $\mathcal{O}(n^2d^n)$ for multiplying polynomials times scalar as well as summing polynomials. This gives a precondition cost of order $\mathcal{O}(n^2d^n)$.

2.5 Regularity tests

A subdivision scheme (cf. Algorithm 1) is able to work when two tests are available: one that identifies empty domains (exclusion test) and one that identifies domains with exactly one zero (inclusion test). If these two tests are negative, a domain cannot be neither included nor excluded so we need to apply further reduction/subdivision steps to it. Nevertheless, if the result of the test is affirmative, then this should be certified to be correct, in order to get a reliable method.

2.5.1 Exclusion test

The bounding functions defined in the previous section provide a fast filter to exclude empty domains. Define $\min \varnothing = \infty$ and $\max \varnothing = 0$.

Corollary 2.8. *If for some $k \in \{1, .., n\}$ and for some $i \in \{1, .., s\}$ it is $\mu_k(f_i) = \infty$ or $\mathcal{M}_k(f_i) = 0$ then the system has no solutions. Also, if $\max_{i=1,..,s}\{\mu_k(f_i)\} > \min_{i=1,..,s}\{\mathcal{M}_k(f_i)\}$ then there can be no solution to the system.*

Proof. For the former statement observe that f_i has no real positive roots, thus the system has no roots. The latter statement means that the reduced domains of each f_i, $i = 1, .., s$ do not intersect, thus there are no solutions. □

We can use interval arithmetic to identify additional empty domains; if the sign of some initial f_i is constant in $I_H = \mathcal{H}(\mathbb{R}^n_{>0})$ then this domain is discarded. We can also simply inspect the coefficients of each $H(f_i)$; if there are no sign changes then the corresponding box contains no solution.

The accuracy of these criteria greatly affects the performance of the algorithm. In particular, the sooner an empty domain is rejected the less subdivisions will take place and the process will terminate faster. We justify that the exclusion criteria will eventually succeed on an empty domain by proving a generalization of Vincent's theorem to the tensor multivariate case.

Theorem 2.9. *Let $f(\boldsymbol{x}) = \sum_{\boldsymbol{i=0}}^{\boldsymbol{d}} c_{\boldsymbol{i}}\, \boldsymbol{x^i}$ be a polynomial with real coefficients, such that it has no (complex) solution with $\Re(z_k) \geq 0$ for $k = 1, .., n$. Then all its coefficients $c_{i_1,..,i_n}$ are of the same sign.*

Proof. We prove the result by induction on n, the number of variables. For $n = 1$, this is the classical Vincent's theorem [3].

Consider now a polynomial

$$f(x_1, x_2) = \sum_{0 \leq i_1 \leq d_1, 0 \leq i_2 \leq d_2} c_{i_1,i_2}\, x_1^{i_1}\, x_2^{i_2}$$

in two variables with no (complex) solution such that $\Re(x_i) \geq 0$ for $i = 1, 2$. We prove the result for $n = 2$, by induction on the degree $d = d_1 + d_2$. The property is obvious for polynomials of degree $d = 0$. Let us assume it for polynomials of degree less than d.

By hypothesis, for any $z_1 \in \mathbb{C}$ with $\Re(z_1) \geq 0$, the univariate polynomial $f(z_1, x_2)$ has no root with $\Re(x_2) \geq 0$. According to Lucas theorem [74], the complex roots of $\partial_{x_2} f(z_1, x_2)$ are in the convex hull of the complex roots of $f(z_1, x_2)$. Thus, there is no root of $\partial_{x_2} f(x_1, x_2)$ with $\Re(x_1) \geq 0$ and $\Re(x_2) \geq 0$. By induction hypothesis, the coefficients of $\partial_{x_2} f(x_1, x_2)$ are of the same sign. We decompose P as

$$f(x_1, x_2) = f(x_1, 0) + f_1(x_1, x_2)$$

where $f_1(x_1, x_2) = \sum_{0 \leq i_1 \leq d_1, 1 \leq i_2 \leq d_2} c_{i_1,i_2}\, x_1^{i_1}\, x_2^{i_2}$ with c_{i_1,i_2} of the same sign, say positive. By Vincent theorem in one variable, as $f(x_1, 0)$ has no root with $\Re(x_1) \geq 0$, the coefficients $c_{i_1,0}$ of $f(x_1, 0)$ are also of the same sign. If this sign is different from the sign of c_{i_1,i_2} for $i_2 \geqslant 1$ (i.e. negative here), then $f(0, x_2)$ has one sign variation in its coefficients list. By Descartes rule, it has one real positive root, which contradicts the hypothesis on f. Thus, all the coefficients have the same sign.

Assume that the property has been proved for polynomials in $n - 1$ variables and let us consider a polynomial $f(\boldsymbol{x}) = \sum_{i=0}^{d} c_i\, \boldsymbol{x}^i$ in n variables with no (complex) solution such that $\Re(x_k) \geq 0$ for $k = 1, .., n$. For any $z_1, .., z_{n-1} \in \mathbb{C}$ with $\Re(z_k) \geq 0$, for $k = 1, .., n-1$, the polynomial $f(z_1, .., z_{n-1}, x_n)$ and $\partial_{x_n} f(z_1, .., z_{n-1}, x_n)$ has no root with $\Re(x_n) \geq 0$. By Lucas theorem and induction hypothesis on the degree, $\partial_{x_n} f(\boldsymbol{x})$ has coefficients of the same sign. We also have $f(x_1, .., x_{n-1}, 0)$ with coefficients of the same sign, by induction hypothesis on the number of variables. If the two signs are different, then $f(0, .., 0, x_n)$ has one sign variation in its coefficients and thus one real positive root, say ζ_n, which cannot be the case, since $(0, .., 0, \zeta_n)$ would yield a real root of f. We deduce that all the coefficients of f are of the same sign.

This completes the inductive proof of the theorem. □

We can reformulate this result for bounded domains, using homography transformations, as follows.

Corollary 2.10. *Let $H(f) = \sum_{i=0}^{\boldsymbol{d}} c_{\boldsymbol{i}}\, \boldsymbol{x}^{\boldsymbol{i}}$ be the representation of f through $\mathcal{H}$ in a box $I_H = [\boldsymbol{u}, \boldsymbol{v}]$. If there is no root $\boldsymbol{z} \in \mathbb{C}^n$ of f such that*

$$\left| z_k - \frac{u_k + v_k}{2} \right| \leq \frac{v_k - u_k}{2}, \text{ for } k = 1, .., n,$$

then all the coefficients $c_{i_1,..,i_n}$ are of the same sign.

That is, if $\|dist\|_\infty(\mathcal{Z}_{\mathbb{C}^n}(f), m) > \dfrac{\delta}{2}$, where m is the center of I_H of size δ, then I_H is excluded by sign conditions.

Proof. The interval $[u_k, v_k]$ is transformed by $\mathcal{H}^{-1}$ into $[0, +\infty]$ and the disk $\left|z_k - \frac{u_k+v_k}{2}\right| \leq \frac{v_k-u_k}{2}$ is transformed into the half complex plane $\Re(z_k) \geq 0$. We deduce that $H(f)$ has no root with $\Re(z_k) \geq 0$, $k = 1, \ldots, n$. By Theorem 2.9, the coefficients of $H(f)$ are of the same sign. □

We deduce that if a domain is far enough from the zero locus of some f_i then it will be excluded, hence redundant empty domains concentrate only in a neighborhood of $\boldsymbol{f} = \boldsymbol{0}$.

The regions which will be excluded during the subdivision algorithm can be quantitatively related to the regions where $\|f(x)\|_\infty$ is large, using the Lipschitz constant of f:

Definition 2.11. *For a system $f = (f_1, \ldots, f_s)$ of polynomials $f_i \in \mathbb{R}[\boldsymbol{x}]$ ($i = 1, \ldots, s$) and a box $I = I_1 \times \cdots \times I_n \subset \mathbb{R}^n$, let*

$$\lambda_I(f) = \max\{\frac{\|f(x) - f(y)\|}{\|x - y\|}; x \neq y \in I_{\mathbb{C}}\}$$

be the Lipschitz constant of f on $I_{\mathbb{C}}$.

By definition, we have $\|f(x) - f(y)\| \leq \lambda_I(f)\|x - y\|$ for all $x, y \in I_{\mathbb{C}}$. It is convenient to define a box in reference to it's "center".

Definition 2.12. *For $x \in \mathbb{R}^n$, $\varepsilon > 0$, we use the symbols:*

- ◇ *The interior of the real hypercube centered at x: $I(x, \varepsilon) = \{y \in \mathbb{R}^n : |y_i - x_i| < \varepsilon,\ i = 1, \ldots, n\}$, i.e. $I = I_1 \times \cdots \times I_n$, with $I_i = I_i(x_i, \varepsilon) \subseteq \mathbb{R}$.*
- ◇ *The complex ball $B_{\mathbb{C}}(x, \varepsilon) = \{y \in \mathbb{C}^n : \|y - x\| < \varepsilon\}$.*
- ◇ *The complex multi-disc $I_{\mathbb{C}}(x, \varepsilon) = \{y \in \mathbb{C}^n : |y_i - x_i| < \varepsilon,\ i = 1, \ldots, n\}$, i.e. $I_{\mathbb{C}} = I_1 \times \cdots \times I_n$, with $I_i = B_{\mathbb{C}}(x_i, \varepsilon) \subseteq \mathbb{C}$.*

Lemma 2.13. *Suppose that $I_{\mathbb{C}} \subset B_{\mathbb{C}}(0,\rho)$ with $\rho \geq 1$ and $f \in \mathbb{R}[\boldsymbol{x}]$ is of degree d, then*

$$\lambda_I(f) \leq \sqrt{2}\, d\, \|f\| \rho^{d-1}.$$

Proof. Let $x, y \in I_{\mathbb{C}} \subset B_{\mathbb{C}}(0,\rho)$. We consider $g(t) := \Re(f(x+t\,(y-x)) - f(x))$. By the intermediate value theorem, there exists $\tau \in [0,1]$ such that

$$|\Re(f(y) - f(x))| = |g'(\tau)| \leq \|D_{y-x}(f)(x + \tau\,(y-x))|$$

The same results applies if we take the imaginary part. By [97, III, Prop. 1 p. 484] we have

$$|f(y) - f(x)| \leq \sqrt{2}\|D_{y-x}(f)(x + \tau\,(y-x))| \leq \sqrt{2}\|D_{y-x}(f)\|\rho^{d-1}$$

since $x + \tau(y-x) \in I_{\mathbb{C}} \subset B_{\mathbb{C}}(0,\rho)$. By [97, III, Lem. 2, p. 485] we deduce that

$$|f(y) - f(x)| \leq \sqrt{2}\, d\, \|f\| \rho^{d-1} \|y - x\|,$$

which concludes the proof of the Lemma. □

Proposition 2.14. *Let H be a homography of $\mathbb{R}^n$ and $\varepsilon > 0$ is such that $|I| < 2\,\varepsilon$, where $I = I_H$. If $\|f(x)\|_\infty > \sqrt{n}\,\lambda_I(f)\,\varepsilon$ then the coefficients of at least one of the functions $H(f_i)$ are of constant sign.*

Proof. Suppose that $\|f(x)\|_\infty = |f_{i_0}(x)|$ and let $z \in \mathbb{C}^n$ be the closest point to x such that $f_{i_0}(z) = 0$. We have

$$|f_{i_0}(x)| = |f_{i_0}(x) - f_{i_0}(z)| \leq \lambda_I(f)\,\|x - z\| \leq \lambda_I(f)\,\sqrt{n}\,\|x - z\|_\infty.$$

and thus there is no root of f_{i_0} in $I_{\mathbb{C}}(x,\varepsilon)$ for $\varepsilon < \frac{|f_{i_0}(x)|}{\sqrt{n}\,\lambda_I(f)}$. By Corollary 2.10, we deduce that the coefficients of $H(f_{i_0})$ are of constant sign. □

To analyze more precisely the subdivision process, we are going to relate the number of boxes which are not removed with some integral geometry invariants [115]:

Definition 2.15. *The tubular neighborhood of size ε of f_i is the set*

$$\tau_\varepsilon(f_i) = \{x \in \mathbb{R}^n \ : \ \exists z \in \mathbb{C}^n,\ f_i(z) = 0,\ \textit{s.t.}\ \|z - x\|_\infty < \varepsilon\}.$$

We bound the number of boxes that are not excluded at each level of the subdivision tree.

Lemma 2.16. *Assume that $I = I_1 \times \cdots \times I_n$ is bounded. There exists $N_f^*(I) \in \mathbb{N}^+$ such that the number of boxes of size $\varepsilon > 0$ kept by the algorithm is less than $N_f^*(I)$ and such that*

$$V(f,\varepsilon) := \text{volume}\,(\cap_{i=1}^s \tau_\varepsilon(f_i) \cap I)) \leq N_f^*(I)\,\varepsilon^n.$$

Proof. Consider a subdivision of the domain I_0 into boxes of size $\varepsilon > 0$. We will bound the number $N_f^\varepsilon(I)$ of boxes in this subdivision that are not rejected by the algorithm. By Corollary 2.10 if a box is not rejected, then we have for all $i = 1, \ldots, s$ $\text{dist}_\infty(\mathcal{Z}_{\mathbb{C}^n}(f_i), m) < \frac{\varepsilon}{2}$, where m is the center of the box. Thus all the points of this box (at distance $< \frac{\varepsilon}{2}$ to m) are at distance $< \varepsilon$ to $\mathcal{Z}_{\mathbb{C}^n}(f_i)$ that is in $\cap_{i=1}^s \tau_\varepsilon(f_i) \cap I$.

To bound $N_f^\varepsilon(I)$, it suffices to estimate the $n-$dimensional volume $V(f, \varepsilon)$, since we have:

$$N_f^\varepsilon(I)\varepsilon^n \leq \text{volume}\left(\cap_{i=1}^s \tau_\varepsilon(f_i) \cap I\right) = V(f, \varepsilon).$$

When ε tends to 0, this volume becomes equivalent to a constant times ε^n. For a square system with simple roots in I, it becomes equivalent to the sum for all real roots ζ in I of the volumes of parallelotopes in n dimensions of height 2ε and edges proportional to the gradients of the polynomials at ζ; More precisely, it is bounded by $\varepsilon^n 2^n \sum_{\zeta \in I, f(\zeta)=0} \frac{\prod_i ||\nabla f_i(\zeta)||}{|J_f(\zeta)|}$. We deduce that there exists an integer $N_f^*(I) \geq 2^n \sum_{\zeta \in I, f(\zeta)=0} \frac{\prod_i ||\nabla f_i(\zeta)||}{|J_f(\zeta)|}$ such that $V(f, \varepsilon) \leq N_f^*(I)\, \varepsilon^n < \infty$. For overdetermined systems, the volume is bounded by a similar expression.

Since $V(f, \varepsilon)\, \varepsilon^{-n}$ has a limit when ε tends to 0, we deduce the existence of the finite constant $N_f^*(I) = \max_{\varepsilon > 0} N_f^\varepsilon(I)$, which concludes the proof of the lemma. □

Notice that preconditioning operations can be used here to improve this bound.

2.5.2 Inclusion tests

We consider two types of inclusion tests (which can easily be combined) and analyze the complexity of the corresponding subdivision process in the following sections.

Miranda's test

We present a first test that discovers common solutions, in a box, or equivalently in $\mathbb{R}_+^n$, through homography. To simplify the statements we assume that the system is square, i.e. $s = n$.

Definition 2.17. *The* lower face *polynomial of f with respect to direction k is* $\text{low}(f, k) = f|_{x_k=0}$. *The* upper face *polynomial of f with respect to k is* $\text{upp}(f, k) = f|_{x_k=\infty} := R_k(f)|_{x_k=0}$.

Lemma 2.18 (Miranda Theorem [77, 109])**.** *If for some permutation* $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, *we have* $\text{sign}(\text{low}(H(f_k), \pi(k)))$ *and* $\text{sign}(\text{upp}(H(f_k), \pi(k)))$ *are constant and opposite for all $k = 1, \ldots, n$, then the equations $(f_1, \ldots, f_n)$ have at least one root in I_H.*

The implementation of Miranda's test can be done efficiently if we compute a $0-1$ matrix with $(i,j)-$th entry 1 iff $\mathrm{sign}(\mathrm{low}(H(f_i),j))$ and $\mathrm{sign}(\mathrm{upp}(H(f_i),j))$ are opposite. Then, Miranda test is satisfied iff there is no zero row and no zero column. To see this observe that the matrix is the sum of a permutation matrix and a $0-1$ matrix iff this permutation satisfies Miranda's test.

We deduce a test that identifies boxes with a single root.

Lemma 2.19. *If in the box I_H, $\det J_f(x)$ is bounded away from 0 and the assumptions of Lemma 2.18 are true for $J_f(x_0)^{-1}f$ where x_0 is the center of I_H, then there is a unique root of $f=(f_1,\ldots,f_n)$ in I_H.*

Proof. By [109] we have that the topological degree $\tau := T(J_f(x_0)^{-1}f, I_H, \underline{0})$ of $J_f(x_0)^{-1}f$ at $\underline{0}$ is equal to ± 1. Suppose $\underline{z}_1,\ldots,\underline{z}_t$ are all the roots of f in I_H. Since by definition $\tau := \sum_{i=1}^{t} \mathrm{sign}\ \det(J_f(x_0)^{-1}\, J_f(\underline{z}_i))$ and $\det(J_f(x_0)^{-1}\, J_f(\underline{z}_i)) > 0$, we have $t=1$. □

Miranda's test can be decided with $\mathcal{O}(n^2)$ (cf. [46]) sign evaluations of the face polynomials, which is done by coefficient-sign inspection, yielding $\mathcal{O}(n^2 d^{n-1})$ operations. Then we need to evaluate $J_{\boldsymbol{f}}$ on I_H and compute it's determinant, $\mathcal{O}(n^2 d^n + n^3)$ operations. Thus overall the inclusion test is decided in $\mathcal{O}(n^2 d^n)$.

Proposition 2.20. *If the real roots of the square system in the initial domain I are simple, then Algorithm 1 stops with boxes isolating the real roots in I_0.*

Proof. If the real roots of $f=(f_1,\ldots,f_n)$ in I_0 are simple, in a small neighborhood of them the Jacobian of f has a constant sign. By the inclusion test, any box included in this neighborhood will be outputted if and only if it contains a single root and has no real roots of the jacobian. Otherwise, it will be further subdivided or rejected. Suppose that the subdivision algorithm does not terminate. Then the size of the boxes kept at each step tends to zero. By Corollary 2.10, these boxes are in the intersection of the tubular neighborhoods $(\cap_{i=1}^{s}\tau_\varepsilon(f_i))\cap\mathbb{R}^n$ for $\varepsilon>0$ the maximal size of the kept boxes. If ε is small enough, these boxes are in a neighborhood of a root in which the Jacobian has a constant size, hence the inclusion test will succeed. By the exclusion criteria, a box domain is not subdivided indefinitely, but is eventually rejected when the coefficients become positive. Thus the algorithm either outputs isolating boxes that contain a real root of the system or rejects empty boxes. This shows, by contradiction, the termination of the subdivision algorithm. □

The α-inclusion test

In this section, we consider another inclusion test, based on α-theory and properties of convergence of Newton's method. This test of existence and unicity of a root in a neighborhood of a point involve the following constants:

Definition 2.21. *For a system $f = (f_1, \ldots, f_n)$ of polynomial equations, we define*

- $\beta_f(x) = \|Df(x)^{-1} f(x)\|$, *where $Df(x)^{-1}$, the inverted Jacobian matrix evaluated at x,*
- $\gamma_f(x) = \sup_{k\geq 2} \|\frac{1}{k!} Df(x)^{-1} D^k f(x)\|^{\frac{1}{k-1}}$, *where $D^k f(x)$, the $k-$th covariant derivative of f,*
- $\alpha_f(x) = \beta_f(x)\, \gamma_f(x)$.

Let $\delta(u) := \frac{1}{4}(1+u-\sqrt{(1+u)^2 - 8\,u})$. We recall here a well-known theorem [97], [13] which is the foundation of the theory:

Theorem 2.22. *If $\alpha_f(x) < \alpha_0 \leq \frac{1}{4}(13-3\sqrt{17}) \sim 0.1577$ then f has a unique zero ζ in the ball $B_{\mathbb{C}}(x, \frac{\delta_0}{\gamma_f(x)})$, with $\delta_0 = \delta(\alpha_0) \leq \frac{2-\sqrt{2}}{2} \sim 0.2929$. Moreover, for each point $z \in B_{\mathbb{C}}(x, \frac{\delta_0}{\gamma_f(x)})$, Newton's method starting from z converges quadratically to ζ.*

This yields the following definition:

Definition 2.23. *A box $I(x, \varepsilon)$ is an α_0-inclusion box if $\alpha_f(x) < \alpha_0$, and $\varepsilon < \sqrt{n}\dfrac{\delta(\alpha_0)}{\gamma_f(x)}$.*

By theorem 2.22, if $I(x, \varepsilon)$ is an α_0-inclusion box then there is a unique root in the ball $B_{\mathbb{C}}(x, \frac{\delta_0}{\gamma_f(x)})$ where $\delta_0 = \delta(\alpha_0)$. Moreover, we have $I(x, \varepsilon) \subset B_{\mathbb{C}}(x, \frac{\delta_0}{\gamma_f(x)})$.

As we will see, the α_0-inclusion boxes that derive from the subdivision process (1) determine regions which isolate the roots of the system $f(x) = 0$. In fact, if we are able to decompose a domain I_0 into a union of boxes which either contain no roots or are α_0-inclusion boxes, then

- each root is in a connected component of the union of the α_0-inclusion boxes, and,
- each connected component of the union of the α_0-inclusion boxes contains a unique root of $f(x) = 0$.

More precisely, if such a connected component is $\cup_{k=1}^{s} I(x^{(k)}, \varepsilon^{(k)})$ with $x^{(k)} \in D_0$ and $\varepsilon^{(k)} > 0$, then by theorem 2.22, there is a unique root $\zeta^{(k)}$ in $B(x^{(k)}, \frac{\delta_0}{\gamma_f(x^{(k)})})$, for $k = 1, \ldots, s$. As the balls $B(x^{(k)}, \frac{\delta_0}{\gamma_f(x^{(k)})})$, $B(x^{(j)}, \frac{\delta_0}{\gamma_f(x^{(j)})})$ of two adjacent α_0-inclusion boxes intersect, we must have

$$\zeta^{(k)} = \zeta^{(j)} \in B(x^{(k)}, \frac{\delta_0}{\gamma_f(x^{(k)})}) \cap B(x^{(j)}, \frac{\delta_0}{\gamma_f(x^{(j)})}).$$

Since the connected component $\cup_{i=1}^{s} I(x^{(k)}, \varepsilon^{(k)})$ is a union of boxes in which every box has at least one adjacent, by recursive application of the above argument we derive that it contains a unique root ζ, which moreover is in $\cap_{k=1}^{s} B(x^{(k)}, \varepsilon^{(k)})$.

2.6 Complexity and continued fractions

In this section we compute an upper bound on the complexity of the algorithm that exploits the continued fraction expansion of the real roots of the system. Hereafter, we call this algorithm MCF (Multivariate Continued Fractions). Since the inclusion test is based on an α-theorem, we assume that the system has *simple* real roots. Since the complexity analysis of the reduction steps of Section 2.4 and the Exclusion-Inclusion test of Section 2.5 would require much more developments, we simplify further the situation and analyze a variant of this algorithm. We assume that two oracles are available. The first one computes, exactly, the partial quotients of the positive real roots of the system, that is the integer part of the coordinates. The second counts exactly the number of real roots of the system inside a hypercube in the open positive orthant, namely $\mathbb{R}^n_+$. In fact, having this second oracle is enough for the realization/implementation of the former, together with bisection. In what follows, we will assume the cost of the first oracle is bounded by $\mathcal{C}_1$, while the cost of the second is bounded by $\mathcal{C}_2$, and we shall derive the total complexity of the algorithm with respect to these parameters. In any case the number of reduction or subdivision steps that we derive is a lower bound on the number of steps that every variant of the algorithm will perform. The next section presents some preliminaries on continued fractions, and then we detail the complexity analysis.

2.6.1 About continued fractions

Our presentation follows closely [104], and we refer the reader to, e.g., [17, 107, 114] for additional details. A *simple (regular) continued fraction* is a (possibly infinite) expression of the form

$$c_0 + \cfrac{1}{c_1 + \cfrac{1}{c_2 + \cdots}} = [c_0, c_1, c_2, \ldots],$$

where the numbers c_i are called *partial quotients*, $c_i \in \mathbb{Z}$ and $c_i \geq 1$ for $i > 0$. Notice that c_0 may have any sign, however, in our real root isolation algorithm $c_0 \geq 0$, without loss of generality. By considering the recurrent relations

$$\begin{array}{lll} P_{-1} = 1, & P_0 = c_0, & P_{n+1} = c_{n+1}\,P_n + P_{n-1}, \\ Q_{-1} = 0, & Q_0 = 1, & Q_{n+1} = c_{n+1}\,Q_n + Q_{n-1}, \end{array}$$

it can be shown by induction that $R_n = \frac{P_n}{Q_n} = [c_0, c_1, \ldots, c_n]$, for $n = 0, 1, 2, \ldots$.

If $\gamma = [c_0, c_1, \ldots]$ then $\gamma = c_0 + \frac{1}{Q_0 Q_1} - \frac{1}{Q_1 Q_2} + \cdots = c_0 + \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{Q_{n-1} Q_n}$ and since this is a series of decreasing alternating terms, it converges to some real number γ. A finite section $R_n = \frac{P_n}{Q_n} = [c_0, c_1, \ldots, c_n]$ is called the $n-$th *convergent* (or

approximant) of γ and the tails $\gamma_{n+1} = [c_{n+1}, c_{n+2}, \ldots]$ are known as its *complete quotients*. That is $\gamma = [c_0, c_1, \ldots, c_n, \gamma_{n+1}]$ for $n = 0, 1, 2, \ldots$. There is a one to one correspondence between the real numbers and the continued fractions, where evidently the finite continued fractions correspond to rational numbers.

It is known that $Q_n \geq F_{n+1}$ and that $F_{n+1} < \phi^n < F_{n+2}$, where F_n is the $n-$th Fibonacci number and $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. Continued fractions are the best rational approximation(for a given denominator size). This is as follows:

$$\frac{1}{Q_n(Q_{n+1} + Q_n)} \leq \left| \gamma - \frac{P_n}{Q_n} \right| \leq \frac{1}{Q_n Q_{n+1}} < \phi^{-2n+1}. \tag{2.7}$$

Let $\gamma = [c_0, c_1, \ldots]$ be the continued fraction expansion of a real number. The Gauss-Kuzmin distribution [17] states that for almost all real numbers γ, that is the set of exceptions has Lebesgue measure zero, the probability for a positive integer δ to appear as an element c_i in the continued fraction expansion of γ is

$$Prob[c_i = \delta] \backsimeq \lg \frac{(\delta+1)^2}{\delta(\delta+2)}, \quad \text{for any fixed } i > 0. \tag{2.8}$$

The Gauss-Kuzmin law induces that we can not bound the mean value of the partial quotients or in other words that the expected value (arithmetic mean) of the partial quotients is diverging, i.e.

$$E[c_i] = \sum_{\delta=1}^{\infty} \delta \, Prob[c_i = \delta] = \infty, \text{ for } i > 0.$$

However, the geometric, as well as the harmonic, mean is not only asymptotically bounded, but is bounded by a constant, for almost all $\gamma \in \mathbb{R}$. For the geometric mean this is the famous Khintchine's constant [58], i.e.

$$\lim_{n\to\infty} \sqrt[n]{\prod_{i=1}^{n} c_i} = \mathcal{K} = 2.685452001...$$

It is not known if $\mathcal{K}$ is a transcendental number. The expected value of the bit size of the partial quotients is a constant for almost all real numbers, when $n \to \infty$ or n sufficiently big [58]. Notice that in (2.8), $i > 0$, thus $\gamma \in \mathbb{R}$ is uniformly distributed in $(0, 1)$. Let $\mathcal{L}(c_i) \triangleq b_i$, then

$$E[b_i] = \mathcal{O}(\lg \mathcal{K}) = \mathcal{O}(1). \tag{2.9}$$

Lévy loosened the assumptions of Khintchine and proved [64] that the distribution also holds for $\gamma \in \mathbb{R}$ with any density function that is Lebesgue measurable.

2.6.2 Complexity results

We denote by σ the upper bound on the bit-size of the partial quotients that appear during the execution of the algorithm.

Lemma 2.24. *The number of reduction and subdivision steps that the algorithm performs is* $\widetilde{\mathcal{O}}(2^n\, n(d+n+\tau)d^{2n-1})$.

Proof. Let $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_n)$ be a real root of the system. It suffices to consider the number of steps needed to isolate the $i-$th coordinate of ζ. We remind the reader that we are working in the positive orthant and we can compute exactly the next partial quotient in each coordinate; in other words a vector $\boldsymbol{l} = (l_1, \ldots, l_n)$, where each l_i, $1 \leq i \leq n$, is the partial quotient of a coordinate of a positive real solution of the system.

Let $k_i(\zeta)$ be the number of steps needed to isolate the i^{th} coordinate of the real root ζ. The analysis extends the one of the univariate case. We may consider the whole process of the subdivision algorithm as a 2^n-ary tree, where at each node we associate a, possible, open hypercube, and to the root of the tree we associate the positive orthant. The leaves of the tree form a partition of the positive orthant, and they contain at most one real root of the system. The number of nodes of the tree correspond to the number of steps that the algorithm performs.

We prune some leaves of the tree to make the counting easier. We prune all the leaves that have siblings that are not leaves. We prune all the leaves that do not contain a real root. Notice that these leaves have at least one sibling that contains a real root, since otherwise the subdivision process would have stopped one step before. All the remaining leaves contain a real root. If there are siblings that are all leaves then we keep arbitrarily one of them. The number of nodes in the original tree is at most 2^n times the number of nodes of the pruned tree.

Now every leave of the pruned tree corresponds to a hypercube that contains exactly one real root, say ζ, of the system. The edges of the hypercube correspond to successive approximations of ζ_i by consecutive approximants. The $k_i(\zeta)$-th approximant is $\frac{P_{k_i(\zeta)}}{Q_{k_i(\zeta)}}$ and following (2.7) should satisfy

$$\left|\frac{P_{k_i(\zeta)}}{Q_{k_i(\zeta)}} - \zeta_i\right| \leq \frac{1}{Q_{k_i(\zeta)}Q_{k_i(\zeta)+1}} < \phi^{-2k_i(\zeta)+1}.$$

In order to isolate ζ_i, it suffices to have

$$\left|\frac{P_{k_i(\zeta)}}{Q_{k_i(\zeta)}} - \zeta_i\right| \leq \Delta_i(\zeta),$$

where $\Delta_i(\zeta)$ is the local separation bound of ζ_i, that is, the smallest distance between ζ_i and all the other $i-$th coordinates of the positive real solutions of the

system. The number of nodes from the hypercube that isolates ζ to the root of the tree is, in the worst case, $k(\zeta) = \max_i k_i(\zeta)$.

Combining the last two equations, we deduce that to achieve the desired approximation, we should have $\phi^{-2k_i(\zeta)+1} \leq \Delta_i(\zeta)$, or $k_i(\zeta) \geq \frac{1}{2} - \frac{1}{2}\lg \Delta_i(\zeta)$. That is, to achieve the desired approximation it suffices to compute $\mathcal{O}(-\frac{1}{2}\lg \Delta_i(\zeta))$ approximants. In other words, from the leaf that corresponds to a hypercube that isolates ζ to the root of the tree there are $\mathcal{O}(-\frac{1}{2}\lg \Delta(\zeta))$ nodes, where $\Delta = \min \Delta_i$.

To compute the total number of steps, i.e. the total number of nodes of the pruned tree, we need to sum over all the real roots that appear at the leaves of the tree; hence

$$\sum_{\zeta \in V} k(\zeta) \leq \frac{1}{2}R - \frac{1}{2}\sum_{\zeta \in V} \lg \Delta(\zeta) = \frac{1}{2}R - \frac{1}{2}\lg \prod_{\zeta \in V} \Delta(\zeta),$$

where $|V| \leq R$, V is the set of positive real roots at the leaves of the pruned tree and R and the total number of positive real roots.

To bound the logarithm of the product, we use $\mathtt{DMM}_n$ [42], i.e. aggregate separation bounds for multivariate, zero-dimensional polynomial systems. It holds

$$\begin{array}{lcl} \prod_{\zeta \in V} \Delta(\zeta) & \geq & 2^{-(3+4\lg n+4n\lg d)d^{2n}}\, 2^{-2n(1+n\lg d+\tau)d^{2n-1}} \\ -\log \prod_{\zeta \in V} \Delta(\zeta) & \leq & (3+4\lg n+4n\lg d)d^{2n} + 2n(1+n\lg d+\tau)d^{2n-1}, \\ -\log \prod_{\zeta \in V} \Delta(\zeta) & = & \widetilde{\mathcal{O}}(nd^{2n} + (n^2+n\tau)d^{2n-1}). \end{array}$$

Taking into account that $R \leq d^n$ we conclude that the total number of nodes of the pruned tree is $\widetilde{\mathcal{O}}(nd^{2n} + n(n+\tau)d^{2n-1})$, and hence the number of steps of the algorithm is $\widetilde{\mathcal{O}}(2^n\, nd^{2n} + 2^n\, n(n+\tau)d^{2n-1})$. □

Proposition 2.25. *The total complexity of the algorithm is* $\widetilde{\mathcal{O}}_B(2^{3n}n^5(n^2+d^2+\tau^2)d^{5n-1}\sigma + (\mathcal{C}_1+\mathcal{C}_2)2^n\, n(d+n+\tau)d^{2n-1})$.

Proof. At each h-th step of algorithm, if there is more than one root of the corresponding system in the positive orthant, (let the cost of estimating this is be $\mathcal{C}_2$), we compute the corresponding partial quotients $l_h = (l_{h,1}, \ldots, l_{h,n})$, where $\mathcal{L}(h_{h,i}) \leq \sigma_h$ (let the cost of this computation be $\mathcal{C}_1$). Then, for each polynomial of the system, f, we perform the shift operation $f(x_1 + l_1, \ldots, x_n + l_n)$, and then we split the positive orthant to 2^n subdomains. Let us estimate the cost of the last two operations.

A shift operation on a polynomial of degree $\leq d$, by a number of bit-size σ, increases the bit-size of the polynomial by an additive factor $nd\sigma$. At the h step of the algorithm, the polynomials of the corresponding system are of bit-size $\mathcal{O}(\tau + nd\sum_{i=1}^{h}\sigma_h)$, and we need to perform a shift operation to all the variables, with number of bit-size σ_{h+1}. The cost of this operation is $\widetilde{\mathcal{O}}_B(nd^n\tau + n^2d^{n+1}\sum_{k=1}^{h+1}\sigma_k)$,

and since we have n polynomials the costs becomes $\widetilde{\mathcal{O}}_B(n^2d^n\tau+n^3d^{n+1}\sum_{k=1}^{h+1}\sigma_k)$, The resulting polynomial has bit-size $\mathcal{O}(\tau+nd\sum_{k=1}^{h+1}\sigma_k)$.

To compute the cost of splitting the domain, we proceed as follows. The cost is bounded by the cost of performing $n2^n$ operations $f(x_1+1,\ldots,x_n+1)$, which in turn is $\widetilde{\mathcal{O}}_B(nd^n\tau+n^2d^{n+1}\sum_{k=1}^{h+1}\sigma_k+n^2d^{n+1})$. So the total cost becomes $\widetilde{\mathcal{O}}_B(2^nn^2d^n\tau+2^nn^3d^{n+1}\sum_{k=1}^{h+1}\sigma_k)$. It remains to bound $\sum_{k=1}^{h+1}\sigma_k$. If σ is a bound on the bit-size of all the partial quotients that appear during and execution of the algorithm, then $\sum_{k=1}^{h+1}\sigma_k=\mathcal{O}(h\sigma)$.

Moreover, $h\leq\#(T)=\widetilde{\mathcal{O}}(2^n\,nd^{2n}+2^n\,n(n+\tau)d^{2n-1})$ (lem. 2.24), and so the cost of each step is $\widetilde{\mathcal{O}}_B(2^{2n}n^4(n+d+\tau)d^{3n}\sigma)$.

Finally, multiplying by the number of steps (lem. 2.24) we get a bound of $\widetilde{\mathcal{O}}_B(2^{3n}n^5(n^2+d^2+\tau^2)d^{5n-1}\sigma)$.

To derive the total complexity we have to take into account that at each step we compute some partial quotients and and we count the number of real root of the system in the positive orthant. Hence the total complexity of the algorithm is $\widetilde{\mathcal{O}}_B(2^{3n}n^5(n^2+d^2+\tau^2)d^{5n-1}\sigma+(\mathcal{C}_1+\mathcal{C}_2)2^n\,n(d+n+\tau)d^{2n-1})$. □

In the univariate case ($n=1$), if we assume that $\sigma=\mathcal{O}(1)$ holds for real algebraic numbers of degree greater that 2, then the cost of $\mathcal{C}_1$ and $\mathcal{C}_2$ is dominated by that of the other steps, that is the splitting operations, and the (average) complexity becomes $\widetilde{\mathcal{O}}_B(d^3\tau)$ and matches the one derived in [104] (without scaling). This assumption is coherent with (2.9).

2.6.3 Further complexity improvements

We can reduce the number of steps that the algorithm performs, and thus improve the total complexity bound of the algorithm, using the same trick as in [104]. The main idea is that the continued fraction expansion of a real root of a polynomial does not depend on the initial computed interval that contains all the roots. Thus, we spread away the roots by scaling the variables of the polynomials of the system by a carefully chosen value.

If we apply the map $(x_1,\ldots,x_n)\mapsto(x_1/2^\ell,\ldots,x_n/2^\ell)$, to the initial polynomials of the system, then the real roots are multiplied by 2^ℓ, and thus their distance increases. The key observation is that the continued fraction expansion of the real roots does not depend on their integer part. Let ζ be any root of the system, and let γ, be the same root after scaling. It holds $\gamma=2^\ell\,\zeta$. From [42] it holds that

$$-\log\prod_{\zeta\in V}\Delta_i(\zeta)\leq(3+4\lg n+4n\lg d)d^{2n}+2n(1+n\lg d+\tau)d^{2n-1},$$

and thus

$$-\log \prod_{\zeta\in V} \Delta_i(\gamma) = -\log 2^{R\ell} \prod_{\zeta\in V} \Delta_i(\zeta)$$
$$\leq (2n\tau d^{2n-1} + 2nd^n)\lg(nd^{2n}) - R\,\ell.$$

If we choose $\ell = \widetilde{\mathcal{O}}(nd^{n-1}(n+d+\tau))$ and assume that $R \leq d^n$ which is the worst case, then $-\log \prod_{\zeta\in V} \Delta_i(\gamma) = \widetilde{\mathcal{O}}(1)$. Thus, following the proof of Lemma 2.24, the number of steps that the algorithm performs is $\mathcal{O}(2^n\, d^n)$.

The bit-size of the scaled polynomials becomes $\widetilde{\mathcal{O}}(n^2d^{n+1} + n^2d^n\tau)$. The total complexity of algorithm is now

$$\widetilde{\mathcal{O}}_B(2^{2n}n^3d^{3n}(n + 2^nd\sigma + n\tau) + 2^nd^n(\mathcal{C}_1 + \mathcal{C}_2)),$$

where σ the maximum bit-size of the partial quotient appear during the execution of the algorithm.

The discussion above combined with Proposition 2.25 lead us to:

Theorem 2.26. *The total complexity of the algorithm is* $\widetilde{\mathcal{O}}_B(2^{2n}n^3d^{3n}(n + 2^nd\sigma + n\tau) + 2^nd^n(\mathcal{C}_1 + \mathcal{C}_2))$.

If we assume that $\sigma = \mathcal{O}(1)$, the bound becomes $\widetilde{\mathcal{O}}_B(d^3\tau)$ when $n = 1$, which agrees with the one proved in [104].

2.7 Complexity and condition number

The complexity analysis presented previously is a worst case analysis in the bit complexity model, which might not reflect the practical behavior of the method. We can gain a better insight by estimating the arithmetic complexity of the algorithm in the real RAM model, using qualitative information attached to a system of polynomial equations, namely its *condition number*. The latter cannot be computed directly from the system, unless we actually know the roots. However, it is nicely related to the distance from the set of systems with degenerate real roots and thus it has a relevant geometric interpretation that will help understand the behavior of the algorithm.

Our analysis relates the complexity of the subdivision algorithm to this geometric invariant attached to the system. It uses tools similar to the ones developed in [24] and [25]. However, we provide a bound which is not exponential but linear in the logarithm of the condition number. This bound is also connected to the complexity results in [80] or [20].

As in the previous section, we will assume that the real roots of the system in the domain of interest are simple. Otherwise the condition number becomes infinite and the bound is trivial.

In the following, we consider a system $f = (f_1, \ldots, f_n)$ of polynomials $f_i \in \mathbb{R}[\boldsymbol{x}]$ of degree $d_i := \deg(f_i)$. We denote by $d = \max\{d_1, \ldots, d_n\}$. We assume that the system f has no multiple root in I_0. We consider a subdivision algorithm based on the exclusion test of Section 2.5.1 and the inclusion test of Section 2.5.2. We assume moreover that there is a constant $0 < \Phi < 1$ such that at each subdivision the size of a box which is kept is at most Φ times the size of its parent box. Consequently, if we apply k subdivision steps, the boxes which are kept are of size Φ^k times the size of the initial box.

We recall here the definitions that will be used in this complexity analysis.

Definition 2.27. *For a system $f = (f_1, \ldots, f_n)$ of polynomials $f_i \in \mathbb{R}[\boldsymbol{x}]$ with $\deg(f_i) = d_i$,*

$$\mu_f(x) := \|f\| \|Df(x)^{-1} \Delta(\sqrt{d_1}\|x\|_1, \ldots, \sqrt{d_n}\|x\|_1)\|,$$

where $\Delta(z_1, \ldots, z_n)$ is the diagonal matrix with entry z_i for the index (i,i) and 0 for the indices (i,j) with $1 \leq i \neq j \leq n$.

For a root $\zeta \in \mathbb{C}^n$, $\mu_f(\zeta)$ is the condition number of the system f at ζ. It measures the distance to the set of systems which are degenerated at ζ. See [13, p. 233]. This distance bounds the size of complex perturbation we can apply on our system, while staying in the safe region of systems with simple roots. However in practice, we usually consider real perturbations. To take into account the distance to real systems which are degenerate, we use the following *real condition number* [25]:

Definition 2.28. *The local condition number at $x \in \mathbb{R}^n$ for the system f is*

$$\kappa_f(x) := \frac{\|f\|}{(\|f\|^2 \mu_f(x)^{-2} + \|f(x)\|_\infty^2)^{\frac{1}{2}}} = \frac{1}{(\mu_f(x)^{-2} + \|f(x)\|_\infty^2 \, \|f\|^{-2})^{\frac{1}{2}}}.$$

This condition number $\kappa_f(x)$ is related to the distance to the set $\Sigma_{\mathbb{R}}(x, \boldsymbol{d})$ of systems of real polynomials $(f_1, \ldots, f_n)$ with $\boldsymbol{d} = (d_1, \ldots, d_n)$, $\deg(f_i) = d_i$ which are singular at x [25]:

$$\kappa_f(x) := \frac{1}{\operatorname{dist}(f, \Sigma_{\mathbb{R}}(x, d))}.$$

In the univariate case $f \in \mathbb{R}[x]$, the value of $\kappa_f(x)$ will be large at the real roots $\zeta \in \mathbb{R}$ of f where $f'(\zeta)$ is small and at local extrema ξ where $|f(\xi)|$ is small. We extend the definition to a domain:

Definition 2.29. *For $I \subset \mathbb{R}^n$, define $\kappa_I(f) := \sup\{k_f(x); x \in I\}$.*

Proposition 2.30. *For all $\sigma \geq \|f\|$ and $\varepsilon < \frac{4\,\alpha_0\,\|f\|^2}{d^{\frac{3}{2}}\,\kappa_f(x)^2\sigma^2}$, we have*

- $\|f(x)\|_\infty > \sigma\,\varepsilon$, *or*
- $\alpha_f(x) < \alpha_0$.

Proof. Let us suppose that $\alpha_f(x) \geq \alpha_0$ and prove that $\|f(x)\|_\infty > \sigma\,\varepsilon$. We consider two cases.

In the case where $\mu_f(x)^{-1} < \|f(x)\|\,\|f\|^{-1}$, we have $\kappa_f(x) > 2^{\frac{1}{2}}\mu_f(x)$. By [97, Proposition 2, p. 467], we have

$$\beta_f(x) \leq \|x\|_1 \mu_f(x) \frac{\|f(x)\|_\infty}{\|f\|},$$

where $\|x\|_1 = (1 + |x_1|^2 + \cdots + |x_n|^2)^{\frac{1}{2}}$. By [97, Proposition 3, p. 468],

$$\gamma_f(x) \leq \frac{1}{2\,\|x\|_1}\mu_f(x) d^{3/2}.$$

We deduce that

$$\frac{1}{4}\,\varepsilon\, d^{\frac{3}{2}} \kappa_f(x)^2 \frac{\sigma^2}{\|f\|^2} < \alpha_0 \leq \alpha_f(x) = \beta_f(x)\,\gamma_f(x) \leq \frac{1}{2}\,\mu_f(x)^2 d^{\frac{3}{2}} \frac{\|f(x)\|_\infty}{\|f\|}$$

which implies that

$$\varepsilon\,\frac{\sigma^2}{\|f\|} < \|f(x)\|_\infty,$$

since $\kappa_f(x) > 2^{\frac{1}{2}}\mu_f(x)$. As $\sigma \geq \|f\|$, we deduce that

$$\varepsilon\,\sigma < \|f(x)\|_\infty.$$

In the case where $\mu_f(x)^{-1} \geq \frac{\|f(x)\|_\infty}{\|f\|}$, we have $\kappa_f(x) \geq 2^{\frac{1}{2}} \frac{\|f\|}{\|f(x)\|_\infty}$ and

$$2\,\varepsilon \frac{\|f\|^2}{\|f(x)\|_\infty^2} \frac{\sigma^2}{\|f\|^2} \leq 2\,\varepsilon\,\kappa_f(x)^2 \frac{\sigma^2}{\|f\|^2} < 4\frac{\alpha_0}{d^{\frac{3}{2}}},$$

which implies that

$$\|f(x)\|_\infty > \sqrt{\frac{d^{\frac{3}{2}}}{2\,\alpha_0}}\sigma\varepsilon^{\frac{1}{2}} > \sigma\,\varepsilon,$$

since $\varepsilon < 1$ and $\frac{d^{\frac{3}{2}}}{2\,\alpha_0} > 1$. □

Let $\alpha_0 = 0.1$ so that $\delta_0 = \delta(\alpha_0) \sim 0.1145$. We bound the complexity of the subdivision algorithm for the exclusion test of section 2.5.1 and the α_0-inclusion test of section 2.5.2.

Theorem 2.31. *The number of arithmetic operations needed to isolate the roots of a polynomial system f with simple roots in $I = I(x, \varepsilon)$, as in Definition 2.21, with $I_{\mathbb{C}}(x, \varepsilon) \subset B_{\mathbb{C}}(x, \rho)$ and $\rho \geq 1$ is in*

$$\mathcal{O}(N_f^*(I)\, d^{n+1}\, (\log(\kappa_I(f)) + d\, \log(\rho) + \log(n)).$$

Proof. By Lemma 2.16, the number of boxes of size ε kept during the subdivision is bounded by $N_f^*(I)$. The number of arithmetic operations is bounded by $N_f^*(I)$ times the cost of a subdivision step times the depth of the subdivision tree. The cost of a subdivision step is in $\mathcal{O}(d^{n+1})$.

We are going to bound the depth of the subdivision tree as follows. We will show that a box of size $\varepsilon < \frac{\alpha_0}{8\, n\, d^{\frac{7}{2}} \kappa_I(f)^2 \rho^{2d-2}}$ either satisfies the exclusion test or is an α_0-inclusion box. By Proposition 2.30, for a box $I(x, \varepsilon)$ with $\varepsilon < \frac{\alpha_0}{8\, n\, d^{\frac{7}{2}} \kappa_I(f)^2 \rho^{2d-2}} < \frac{4\, \alpha_0 \|f\|^2}{d^{\frac{3}{2}} \kappa_I(f)^2 (2\, n\, d^2\, \rho^{2d-2} \|f\|^2)}$, we have

- ◇ either $\|f(x)\| > \sqrt{2\, n} d \|f\| \rho^{d-1} \varepsilon$, which implies by Lemma 2.13 and Proposition 2.14 that the box $I(x, \varepsilon)$ satisfies the exclusion test;
- ◇ or $\alpha_f(x) < \alpha_0$ and by Theorem 2.22 there is a unique root ζ of $f(\boldsymbol{x}) = 0$ in $B(x, \frac{\delta_0}{\gamma_f(x)})$.

To prove that in the latter case, the box $I(x, \varepsilon)$ is an α_0-inclusion box, we need to check that

$$\frac{\delta_0}{\gamma_f(x)} \geq n^{\frac{1}{2}} \varepsilon.$$

By [97, Proposition 2, p. 476], the following holds:

$$\frac{1}{\gamma_f(x)} \geq \frac{\nu_0}{\gamma_f(\zeta)},$$

where $\nu_0 = (2\delta_0^2 - 4\delta_0 + 1)(1 - \delta_0) \sim 0.5016$. By [97, Proposition 3, p. 468], we have

$$\frac{1}{\gamma_f(\zeta)} \geq \frac{2}{d^{\frac{3}{2}} \mu_f(\zeta)} \geq \frac{2}{d^{\frac{3}{2}} \kappa_I(f)},$$

since ζ is a root of the system and thus $\mu_f(\zeta) = \kappa_f(\zeta) \leq \kappa_I(f)$. This implies that

$$\frac{\delta_0}{\gamma_f(x)} \geq \frac{2\delta_0\nu_0}{d^{\frac{3}{2}} \kappa_I(f)}.$$

As $\varepsilon < \frac{\alpha_0}{8\, n d^{\frac{7}{2}} \kappa_I(f)^2 \rho^{2d}}$ and $\varepsilon < 1$, we have $\frac{1}{\kappa_I(f)} > \sqrt{\frac{8\, \rho^{2d}\, d^{\frac{7}{2}}\, n\, \varepsilon}{\alpha_0}} > \frac{2\sqrt{2} \rho^d\, d^{\frac{7}{4}}\, n^{\frac{1}{2}}}{\sqrt{\alpha_0}} \varepsilon$. We deduce that

$$\frac{\delta_0}{\gamma_f(x)} > \frac{4\sqrt{2}\, \rho^d\, \delta_0 \nu_0}{\sqrt{\alpha_0}} d^{\frac{1}{4}} n^{\frac{1}{2}} \varepsilon > n^{\frac{1}{2}} \varepsilon,$$

since $\rho \geq 1$, $d \geq 1$ and $\frac{4\sqrt{2}\delta_0\nu_0}{\sqrt{\alpha_0}} > 1$. This proves that the box $I(x, \varepsilon)$ is an α_0-inclusion box. Therefore the subdivision step must stop before this precision, which gives a bound of order $\mathcal{O}(\log(\kappa_I(f) + d\log(\rho))$ for the depth of the subdivision tree. □

2.8 Implementation and experimentation

We have implemented the algorithm in the C++ library `realroot` of MATHEMAGIX [106], which is an open source effort that provides fundamental algebraic operations such as algebraic number manipulation tools, different types of univariate and multivariate polynomial root solvers, resultant and GCD computations, etc [106].

The polynomials are internally represented as a vector of coefficients along with some additional data, such as a variable dictionary and the degree of the polynomial in every variable. This allows us to map the tensor of coefficients to the one-dimensional memory. The univariate solver that is used is the continued fraction solver; this is essentially the same algorithm with a different inclusion criterion, namely Descartes' rule. The same data structures is used to store the univariate polynomials, and the same shift/contraction routines. The univariate solver outputs a lower bound on the smallest positive root, as a result of a depth-first strategy during the subdivision algorithm. Our code is templated and support different types of coefficients. It can use the integer arithmetic of GMP, since long integers appear as the box size decreases.

The user needs to provide, together with the system to solve, a threshold parameter $\varepsilon > 0$. This is the minimum width that a box can reach. By using smaller values for this parameter one can obtain a greater precision of the roots. In practice, the bottleneck is the isolation part: once the roots are isolated, a predefined precision can be acquired fast by a bisection iteration on the isolation box.

The threshold parameter also serves in the event of existence of multiple roots. In this case, the algorithm fails to certify the root in the box, thus subdivision continues around the root up to threshold size, and any undecided boxes are marked as potential roots. For roots of small multiplicity (i.e. double roots) the output is still correct most of the time. The subdivision-tree depth is in this case proportional to $-\log\varepsilon$, which should be ultimately chosen equal to known separation bounds of the roots [42]. In the next chapter, we shall introduce a method that can cope with this deficiency.

The following four examples demonstrate the output of our implementation, which we visualize using AXEL*.

*`http://axel.inria.fr`

First, we consider the system $f_1 = f_2 = 0$ (Σ_1), where $f_1 = x^2 + y^2 - xy - 1$, and $f_2 = 10xy - 4$. We are looking for the real solutions in the domain $I = [-2, 3] \times [-2, 3]$, which is mapped to $\mathbb{R}^2_+$, by an initial transformation. The isolating boxes of the real roots can be seen in Fig. 2.3.

For the test-system (Σ_2), We multiply f_1 and f_2 by quadratic components, hence we obtain

$$(\Sigma_2) \begin{cases} f_1 = x^4 + 2x^2y^2 - 2x^2 + y^4 - 2y^2 - x^3y - xy^3 + xy + 1 \\ f_2 = 20x^3y - 10x^2y^2 - 10xy^3 - 8x^2 + 4xy + 4y^2 \end{cases}$$

The isolating boxes of this system could be seen in Fig. 2.4. Notice, that the size of the isolation boxes that are returned in this case is considerably smaller.

We turn now to a system with multiple roots, to demonstrate the behavior of the algorithm in this case. The following system has 7 simple roots, a double root at $(1, 0)$ and a root of multiplicity 12 at $(0, 0)$.

$$(\Sigma_3) \begin{cases} f_1 = -(x + y - y^2)(x - y + y^2)(x^2 + x - y)(x^2 - x - y) \\ f_2 = x^4 + 2x^2y^2 + y^4 + 3x^2y - y^3 \end{cases}$$

We can see in Fig. 2.4 that the simple roots have been recognized. A box is returned that contains the double root. This could not neither be confirmed nor excluded by the algorithm, thus it is marked as potential root. The size of this box attains the threshold that we gave in the input, here $\varepsilon = 0.001$. For the root of higher multiplicity, we can see after zooming in that there is a collection of boxes around $(0, 0)$ that are marked as potential roots. In view of this behaviour, we realize the need for a test that can treat a multiple point. This shall be the objective of Chapter 3.

Consider the system (Σ_4), consisting of $f_1 = x^4 - 2x^2 - y^4 + 1$ and f_2, which is a polynomial of bi-degree $(8, 8)$. The output of the algorithm, that is, the isolating boxes of the real roots can be seen in Fig. 2.4. One important observation is that the isolating boxes *are not* squares, which verifies the adaptive nature of the proposed algorithm.

We provide execution details on these experiments in Table 5.1. Several optimizations can be applied to our code, but the results already indicate that our approach competes well with the Bernstein case.

We compared our implementation with the Bernstein solver of [80] on a number of bi-variate systems, and we present the times in milliseconds in Table 2.2. The tests were run on the same machine and the timings are rounded averages over 10 executions. When using machine integers for representing the polynomials, the Bernstein solver is proved faster from MCF, but the timings for both solvers are of the same order. If we use GMP integers then our algorithm is 10-20 times slower than the Bernstein solver; this difference is expected since GMP integers ought to be slower than machine numbers.

System	Domain	Iters.	Subdivs.	Sols.	Excluded
Σ_1	$[-2,3]^2$	53	26	4	25
Σ_2	$[-2,3]^2$	263	131	12	126
Σ_3	$[-2,3]^2$	335	167	8	160
Σ_4	$[-3,3]^2$	1097	548	16	533

Table 2.1: Execution data for Σ_1, Σ_2, Σ_3, Σ_4.

Degrees in (x,y)	Domain	MCF(integer)	MCF(GMP)	Bernstein
$(2,1),(3,1)$	$[0,2]\times[0,2]$	20	110	2
$(4,4),(2,1)$	$[0,1]\times[0,1]$	70	280	30
$(6,6),(3,2)$	$[-2,2]\times[-2,2]$	10	200	10
$(4,3),(7,6)$	$[-5,5]\times[-5,5]$	110	600	20
$(8,8),(6,7)$	$[0,10]\times[0,10]$	110	540	20
$(8,8),(6,7)$	$[-2,2]\times[-2,2]$	960	8820	490
$(16,16),(12,15)$	$[0,10]\times[0,10]$	460	6550	320

Table 2.2: Running times in ms for our implementation (MCF) and the Bernstein solver [80].

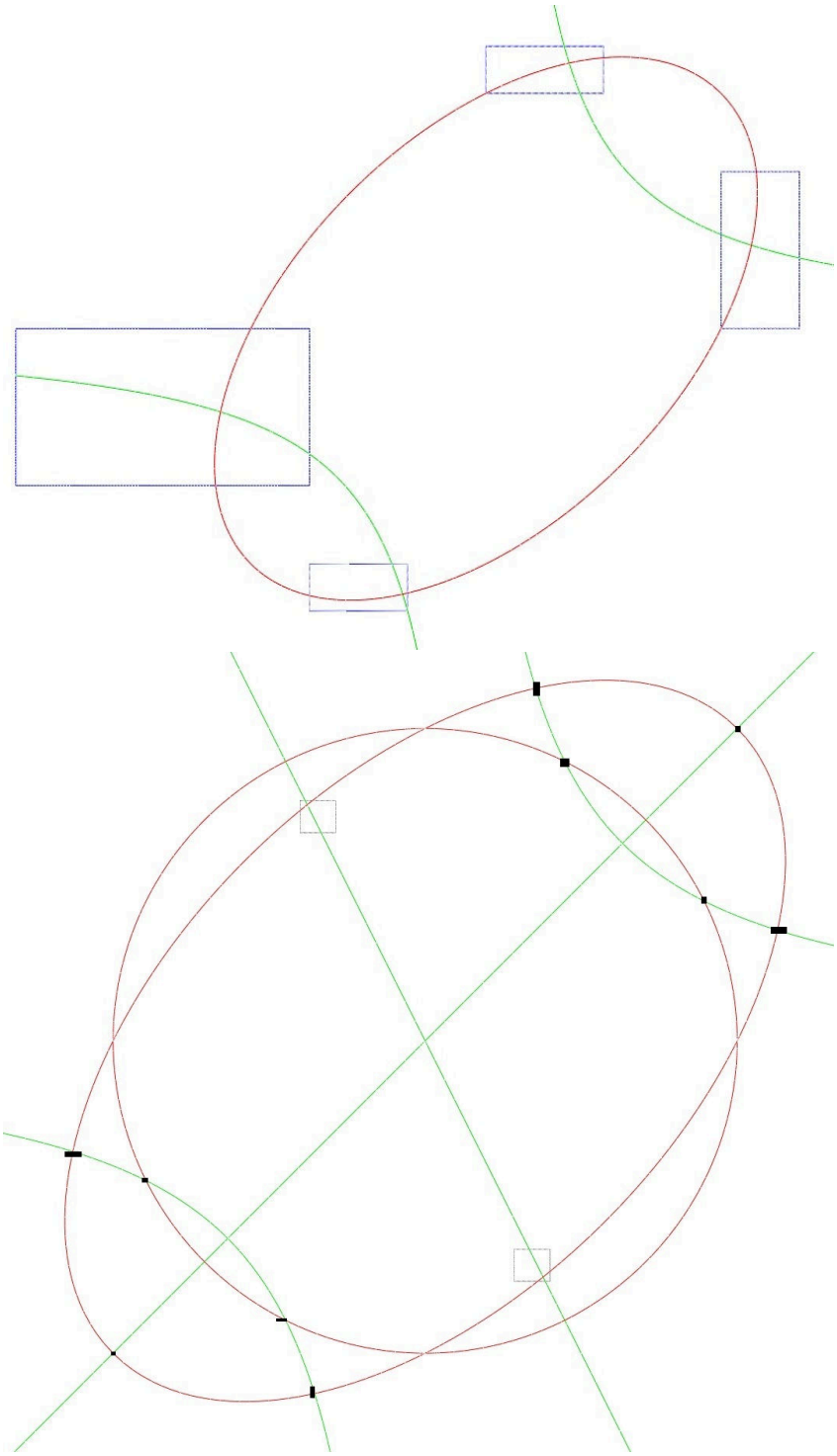

Figure 2.3: Isolating boxes of the real roots of Left: (Σ_1), Right:(Σ_2).

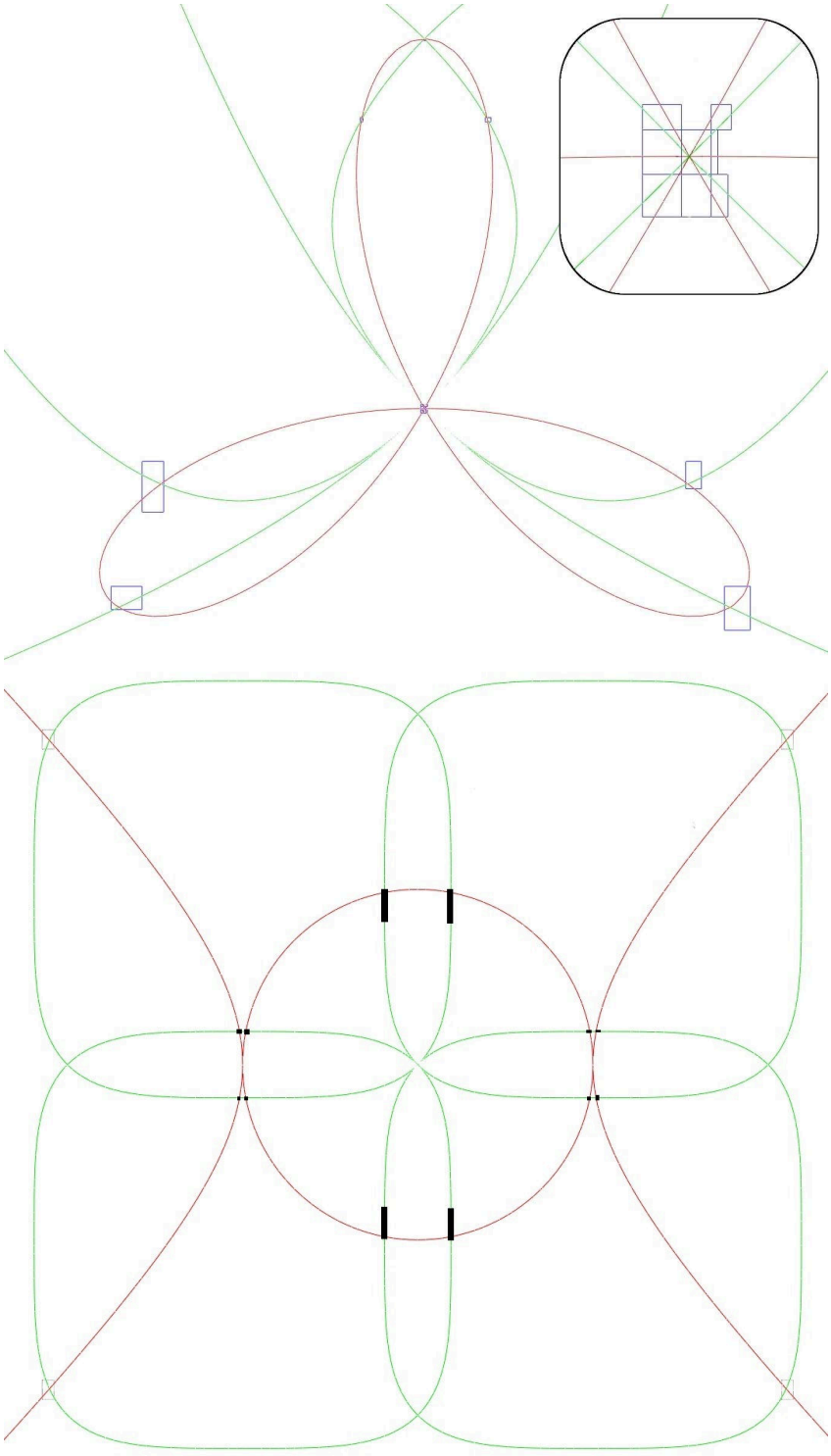

Figure 2.4: Isolating boxes of the real roots of the system Left: (Σ_3), detail close to a multiple point, Right: (Σ_4).

CHAPTER 3

On the treatment of singular isolated roots

Contents

A difficulty encountered in the previous chapter was the failure of the MCF algorithm to answer if a box contains one single multiple root. In this chapter we use the concept of local bases at a multiple point in order to undertake this task. We present a method that can compute this local basis, notably starting from approximate data. Then, interval arithmetic and deflation are used to derive the missing test for singular root certification inside a real a domain. We give some notation in Section 3.2, dual bases in Section 3.3, a mehtod to do deflation in Section 3.4, and the certification test in Section 3.5. In Section 3.6, we present applications to the topology analysis of curves, and experimentation follows in the last section. The proposed method has appeared in [71].

3.1 Introduction

A main challenge in algebraic and geometric computing is singular point identification and treatment. Such problems naturally occur when computing the topology of implicit curves or surfaces [2], or the intersection of parametric surfaces. Isolated singularities can also be seen from the point of view of computational invariant theory [78], as objects having a local structure that describes their geometric characteristics. When algebraic representations are used, singular points appear as solutions of polynomial systems.

In the framwork of subdivision algorithms, a numerical approximation or a box of isolation is computed to identify every real root of the polynomial system. But we often need to improve the numerical approximation of the roots. Numerical methods such as Newton's iteration can be used to improve the quality of the approximation, provided that we have a simple root. In the presence of a multiple root, the difficulties are significantly increasing. The numerical approximation can be of very bad quality, and the methods used to compute this approximation are converging slowly (or not converging). The situation in practical problems, as encountered in CAGD for instance, is even worse, since the coefficients of the input equations are known, with some incertitude. Computing multiple roots of approximate polynomial systems seems to be an ill-posed problem, since changing slightly the coefficients may transform a multiple root into a cluster of simple roots (or even make it disappear).

To tackle this problem, we adopt the following strategy. We try to find a (small) perturbation of the input system such that the root we compute is an exact multiple root of this perturbed system. In this way, we identify the multiplicity structure and we are able to setup deflation techniques which restore the quadratic convergence of the Newton system. The certification of the multiple root is also possible on the symbolically perturbed system by applying a fixed point theorem, based e.g. on interval arithmetic [90] or α-theorems ([47] and references therein).

In order to develop Newton-type methods that converge to multiple roots, deflation techniques which consist in adding new equations in order to reduce the multiplicity have already been considered. In [82], by applying a triangulation preprocessing step on the Jacobian matrix at the approximate root, minors of the Jacobian matrix are added to the system to reduce the multiplicity.

In [63], a presentation of the ideal in a triangular form in a good position and derivations with respect to the leading variables are used to iteratively reduce the multiplicity. This process is applied for p-adic lifting with exact computation.

In [65, 66], instead of triangulating the Jacobian matrix, the number of variables is doubled and new equations are introduced, which are linear in the new variables. They describe the kernel of the Jacobian matrix at the multiple root.

In [26], this construction is related to the construction of the inverse system.

The dialytic method of F.S. Macaulay [69] is revisited for this purpose. These deflation methods are applied iteratively until the root becomes regular, doubling each time the number of variables.

More recent algorithms for the construction of inverse systems are described e.g. in [75], reducing the size of the intermediate linear systems (and exploited in [99]), or in [79] using an integration method.

In [87], a minimization approach is used to reduce the value of the equations and their derivatives at the approximate root, assuming a basis of the inverse system is known.

In [110], the inverse system is constructed via Macaulay's method; tables of multiplications are deduced and their eigenvalues are used to improve the approximated root. They show that the convergence is quadratic at the multiple root.

Verification of multiple roots of (approximate) polynomial equations is a difficult task. The approach proposed in [90] consists of introducing perturbation parameters and to certifying the multiple root of nearby system by using a fixed point theorem, based on interval arithmetic. It applies only to cases where the Jacobian has corank equal to 1.

Our approach is based on certain differential invariants of the singular point, that form a basis of functionals, first studied by [69].

In preparation for the multivariate case, we review some techniques used to treat singular zeros of univariate polynomials, and we present our method on a univariate instance.

Let $g(x) \in \mathbb{R}[x]$ be a polynomial which attains at $x = 0$ a root of multiplicity $\mu > 1$. The latter is defined as the positive integer μ such that $d^\mu g(0) \neq 0$ whereas $g(0) = dg(0) = \cdots = d^{\mu-1}g(0) = 0$. Here we denote $d^k g(x) = \frac{d^k}{dx^k} g(x)/k!$ the normalized $k-$th order derivative with respect to x.

We see that $\mathscr{D}_0 = \langle 1, d, \ldots, d^{\mu-1} \rangle$ is the maximal space of differentials which is stable under derivation, that vanish when applied to members of $\mathcal{Q}_0$, the $\langle x \rangle-$primary component of $\langle g \rangle$ at $x = 0$.

Consider now the symbolically perturbed equation

$$f_1(x, \boldsymbol{\varepsilon}) = g(x) + \varepsilon_1 + \varepsilon_2 x + \cdots + \varepsilon_{\mu-1} x^{\mu-2} \tag{3.1}$$

and apply every basis element of $\mathscr{D}_0$ to arrive to the new system $\boldsymbol{f}(x, \boldsymbol{\varepsilon}) = \left(f_1, df_1, \ldots, d^{\mu-1} f_1 \right)$ in $\mu - 1$ variables. The $i-$th equation is $f_i = d^{i-1} f_1 = d^{i-1} g + \sum_{k=i}^{\mu-1} \binom{k-1}{i-1} x^{k-i} \varepsilon_k$, i.e linear in $\boldsymbol{\varepsilon}$, the last one being $f_\mu = d^{\mu-1} g(x)$. This system deflates the root, as we see that the determinant of its Jacobian matrix at

$(0, \mathbf{0})$ is

$$\det J_{\boldsymbol{f}}(0,\mathbf{0}) = \left|\begin{array}{c|ccc} \frac{d}{dx}f_1 & 1 & & 0 \\ \vdots & & \ddots & \\ \frac{d}{dx}f_{\mu-1} & 0 & & 1 \\ \hline \frac{d}{dx}f_{\mu} & & 0 & \end{array}\right| \begin{array}{l} = -df_\mu(0) \\ = -\mu d^\mu g(0) \neq 0. \end{array}$$

Now suppose that ζ^* is an approximate zero, close to $x = \zeta$. We can still compute $\mathcal{D}_\zeta$ by evaluating $g(x)$ and the derivatives up to a threshold relative to the error in ζ^*. Then we can form (3.1) and use verification techniques to certify the root. Checking that the Newton operator is contracting shows the existence and unicity of a multiple root in a neighborhood of the input data. We are going to extend this approach, described in [90], to multi-dimensional isolated multiple roots.

The proposed algorithm, that shall be presented in detail in the sequel, consists of the following steps:

(a) Compute a basis for the dual space and of the local quotient ring at a given (approximate) singular point.

(b) Deflate the system by augmenting it with new equations derived from the dual basis, introducing adequate perturbation terms.

(c) Certify the singular point and its multiplicity structure for the perturbed system checking the contraction property of Newton iteration (e.g. via interval arithmetic).

In step (a), a dual basis at the singular point is computed by means of linear algebra, based on the integration approach of [79]. We describe an improvement of this method, which yields directly a triangular dual basis with no redundant computation. This method has the advantage to reduce significantly the size of the linear systems to solve at each step, compared to Macaulay's type methods [26, 65, 66, 69]. In the case of an approximate singular point, errors are introduced in the coefficients of the basis elements. Yet a successful computation is feasible. In particular, the support of the basis elements is revealed by this approximate process.

In the deflation step (b), new equations and new variables are introduced in order to arrive to a new polynomial system where the singularity is obviated. The new variables correspond to perturbations of the initial equations along specific polynomials, which form a dual counterpart to the basis of the dual space. One of the deflated systems that we compute from the dual system is a square $n \times n$ system with a simple root. This improves the deflation techniques described in [26, 65, 66], which require additional variables and possibly several deflation steps. New variables are introduced only in the case where we want to certify the multiplicity structure. The perturbation techniques that we use extend the approach of [90] to general cases where the co-rank of the Jacobian matrix could be bigger

than one. The verification step (c) is mostly a contraction condition, using e.g. techniques as in [90]. This step acts on the (approximate) deflated system, since verifying a simple solution of the deflated system induces a certificate of an exact singular point of (a nearby to) the initial system.

3.2 Preliminary considerations

We denote by $R = \mathbb{R}[\boldsymbol{x}]$ a polynomial ring over the field $\mathbb{R}$ of characteristic zero. Also, the *dual ring* R^* is the space of linear functionals $\Lambda : R \to \mathbb{R}$. It is commonly identified as the space of formal series $\mathbb{R}[[\boldsymbol{\partial}]]$ where $\boldsymbol{\partial} = (\partial_1, \ldots, \partial_n)$ are formal variables. Thus we view dual elements as formal series in differential operators at a point $\boldsymbol{\zeta} \in \mathbb{R}^n$. To specify that we use the point $\boldsymbol{\zeta}$, we also denote these differentials $\boldsymbol{\partial}_{\boldsymbol{\zeta}}$. When applying $\Lambda(\boldsymbol{\partial}_{\boldsymbol{\zeta}}) \in \mathbb{R}[[\boldsymbol{\partial}_{\boldsymbol{\zeta}}]]$ to a polynomial $g(\boldsymbol{x}) \in R$ we will denote by $\Lambda^{\boldsymbol{\zeta}}[g] = \Lambda^{\boldsymbol{\zeta}} g = \Lambda(\boldsymbol{\partial}_{\boldsymbol{\zeta}})[g(\boldsymbol{x})]$ the operation

$$\Lambda^{\boldsymbol{\zeta}}[g] = \sum_{\boldsymbol{\alpha} \in \mathbb{N}^n} \frac{\lambda_{\boldsymbol{\alpha}}}{\alpha_1! \cdots \alpha_n!} \cdot \frac{d^{|\boldsymbol{\alpha}|} g}{dx_1^{\alpha_1} \cdots dx_n^{\alpha_n}}(\boldsymbol{\zeta}), \tag{3.2}$$

for $\Lambda(\boldsymbol{\partial}_\zeta) = \sum \lambda_{\boldsymbol{\alpha}} \frac{\mathbf{1}}{\boldsymbol{\alpha}!} \boldsymbol{\partial}_\zeta^{\boldsymbol{\alpha}} \in \mathbb{R}[[\boldsymbol{\partial}_\zeta]]$. Extending this definition to an ordered set $\mathcal{D} = (\Lambda_1, \ldots, \Lambda_\mu) \in \mathbb{K}[[\boldsymbol{\partial}]]^\mu$, we shall denote $\mathcal{D}^{\boldsymbol{\zeta}}[g] = (\Lambda_1^{\boldsymbol{\zeta}} g, \ldots, \Lambda_\mu^{\boldsymbol{\zeta}} g)$. In some cases, it is convenient to use normalized differentials instead of $\boldsymbol{\partial}$: for any $\boldsymbol{\alpha} \in \mathbb{N}^n$, we denote $\boldsymbol{d}_{\boldsymbol{\zeta}}^{\boldsymbol{\alpha}} = \frac{1}{\boldsymbol{\alpha}!} \boldsymbol{\partial}_{\boldsymbol{\zeta}}^{\boldsymbol{\alpha}}$. When $\boldsymbol{\zeta} = \mathbf{0}$, we have $\boldsymbol{d}_{\mathbf{0}}^{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\beta}} = 1$ if $\boldsymbol{\alpha} = \boldsymbol{\beta}$ and 0 otherwise. More generally, $(\boldsymbol{d}_{\boldsymbol{\zeta}}^{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \in \mathbb{N}^n}$ is the dual basis of $((\boldsymbol{x} - \boldsymbol{\zeta})^{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \in \mathbb{N}^n}$.

For $\Lambda \in R^*$ and $p \in R$, let $p \cdot \Lambda : q \mapsto \Lambda(p\, q)$. We check that

$$(x_i - \zeta_i) \cdot \boldsymbol{\partial}_\zeta^{\boldsymbol{\alpha}} = \frac{d}{d\partial_{i,\zeta}} (\boldsymbol{\partial}_\zeta^{\boldsymbol{\alpha}}). \tag{3.3}$$

This property shall be useful in the sequel.

3.2.1 Isolated points and differentials

Let $\mathcal{I} = \langle f_1, \ldots, f_s \rangle$ be an ideal of R, $\boldsymbol{\zeta} \in \mathbb{R}^n$ a root of $\boldsymbol{f}$ and $m_{\boldsymbol{\zeta}} = \langle x_1 - \zeta_1, \ldots, x_n - \zeta_n \rangle$ the maximal ideal at $\boldsymbol{\zeta}$. Suppose that $\boldsymbol{\zeta}$ is an isolated root of $\boldsymbol{f}$, then a minimal primary decomposition of $\mathcal{I} = \bigcap_{\mathcal{Q}\,\text{prim.} \supset \mathcal{I}} \mathcal{Q}$ contains a primary component $\mathcal{Q}_{\boldsymbol{\zeta}}$ such that $\sqrt{\mathcal{Q}_{\boldsymbol{\zeta}}} = m_{\boldsymbol{\zeta}}$ and $\sqrt{\mathcal{Q}'} \not\subset m_{\boldsymbol{\zeta}}$ for the other primary components $\mathcal{Q}'$ associated to $\mathcal{I}$ [5].

As $\sqrt{\mathcal{Q}_\zeta} = m_{\boldsymbol{\zeta}}$, $R/\mathcal{Q}_{\boldsymbol{\zeta}}$ is a finite dimensional vector space. The multiplicity $\mu_{\boldsymbol{\zeta}}$ of $\boldsymbol{\zeta}$ is defined as the dimension of $R/\mathcal{Q}_{\boldsymbol{\zeta}}$. A point of multiplicity one is called regular point, or simple root, otherwise we say that $\boldsymbol{\zeta}$ is a singular isolated point, or multiple root of $\boldsymbol{f}$. In the latter case we have $J_{\boldsymbol{f}}(\boldsymbol{\zeta}) = 0$.

We can now define the dual space of an ideal.

Definition 3.1. *The* dual space *of $\mathcal{I}$ is the subspace of elements of $\mathbb{R}[[\boldsymbol{\partial}_\zeta]]$ that vanish on all the elements of $\mathcal{I}$. It is also called the orthogonal of $\mathcal{I}$ and denoted by $\mathcal{I}^\perp$.*

The dual space is known to be isomorphic to the quotient $R/\mathcal{I}$. Consider now the *orthogonal* of $\mathcal{Q}_\zeta$, i.e. the subspace $\mathscr{D}_\zeta$ of elements of R^* that vanish on members of $\mathcal{Q}_\zeta$, namely

$$\mathcal{Q}_\zeta^\perp = \mathscr{D}_\zeta = \{\Lambda \in R^* \,:\, \Lambda^\zeta[p] = 0,\, \forall p \in \mathcal{Q}_\zeta\}.$$

The following is an essential property that allows extraction of the local structure $\mathscr{D}_\zeta$ directly from the "global" ideal $\mathcal{I} = \langle \boldsymbol{f} \rangle$, notably by matrix methods outlined in Sect. 3.3.

Proposition 3.2 ([79, Th. 8])**.** *For any isolated point $\boldsymbol{\zeta} \in \mathbb{R}$ of $\boldsymbol{f}$, we have $\mathcal{I}^\perp \cap \mathbb{R}[\boldsymbol{\partial}_\zeta] = \mathscr{D}_\zeta$.*

In other words, we can identify $\mathscr{D}_\zeta = \mathcal{Q}_\zeta^\perp$ with the space of polynomial differential operators that vanish at $\boldsymbol{\zeta}$ on every element of $\mathcal{I}$. Also note that $\mathscr{D}_\zeta^\perp = \mathcal{Q}_\zeta$.

The space $\mathscr{D}_\zeta$ has dimension μ_ζ, the multiplicity at $\boldsymbol{\zeta}$. As the variables $(x_i - \zeta_i)$ act on R^* as derivations (see (3.3)), $\mathscr{D}_\zeta$ is a space of differential polynomials in $\boldsymbol{\partial}_\zeta$, which is stable under derivation. This property will be used explicitly in constructing $\mathscr{D}_\zeta$ (Sect. 3.3).

Definition 3.3. *The* nilindex *of $\mathcal{Q}_\zeta$ is the maximal integer $N \in \mathbb{N}$ s.t. $m_\zeta^N \not\subset \mathcal{Q}_\zeta$.*

It is directly seen that the maximal order of elements in $\mathscr{D}_\zeta$ is equal to N, also known as the *depth* of the space.

3.2.2 Quotient ring and dual structure

In this section we explore the relation between the dual ring and the quotient $R/\mathcal{Q}_\zeta$ where $\mathcal{Q}_\zeta$ is the primary component of the isolated point $\boldsymbol{\zeta}$. We show how to extract a basis of this quotient ring from the support of the elements of $\mathscr{D}_\zeta$ and how $\mathscr{D}_\zeta$ can be used to reduce any polynomial modulo $\mathcal{Q}_\zeta$.

It is convenient in terms of notation to make the assumption $\boldsymbol{\zeta} = \mathbf{0}$. This saves some indices, while it poses no constraint (since it implies a linear change of coordinates), and shall be adopted hereafter and in the next section.

Let $\mathrm{supp}\mathscr{D}_\mathbf{0}$ be the set of exponents of monomials appearing in $\mathscr{D}_\mathbf{0}$, with a non-zero coefficient. These are of degree at most N, the nilindex of $\mathcal{Q}_\mathbf{0}$. Since $(\forall \Lambda \in \mathscr{D}_\mathbf{0},\, \Lambda^\mathbf{0}[p] = 0)$ iff $p \in \mathscr{D}_\mathbf{0}^\perp = \mathcal{Q}_\mathbf{0}$, we derive that $\mathrm{supp}\,\mathscr{D}_\mathbf{0} = \{\boldsymbol{\alpha} \,:\, \boldsymbol{x}^{\boldsymbol{\alpha}} \notin \mathcal{Q}_\mathbf{0}\}$. In particular, we can find a basis of $R/\mathcal{Q}_\mathbf{0}$ between the monomials

$\{\boldsymbol{x}^{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \operatorname{supp} \mathscr{D}\}$. This is a finite set of monomials, since their degree is bounded by the nilindex of $\mathcal{Q}_{\mathbf{0}}$.

Given a monomial basis $\mathcal{B} = (\boldsymbol{x}^{\boldsymbol{\beta}_i})_{i=1,\dots,\mu}$ of $R/\mathcal{Q}_{\mathbf{0}}$ and, for all monomials $\boldsymbol{x}^{\boldsymbol{\gamma}_j} \notin \mathcal{Q}_{\mathbf{0}}$, $j = 1, \dots, s-\mu$, $s = \#\operatorname{supp} \mathscr{D}_{\mathbf{0}}$, with $\boldsymbol{x}^{\boldsymbol{\gamma}_j} \notin \mathcal{B}$, the expression (normal form)

$$x^{\boldsymbol{\gamma}_j} = \sum_{i=1}^{\mu} \lambda_{ij} \boldsymbol{x}^{\boldsymbol{\beta}_i} \mod \mathcal{Q}_{\mathbf{0}} \tag{3.4}$$

of $x^{\boldsymbol{\gamma}_j}$ in the basis $\mathcal{B}$ then the dual elements [79, Prop. 13]

$$\Lambda_i(\boldsymbol{d}) = \boldsymbol{d}^{\boldsymbol{\beta}_i} + \sum_{j=1}^{s-\mu} \lambda_{ij} \boldsymbol{d}^{\boldsymbol{\gamma}_j}, \tag{3.5}$$

for $i = 1, \dots, \mu$ form a basis of $\mathscr{D}_{\mathbf{0}}$. We give a proof of this fact in the following lemma.

Lemma 3.4. *The set of elements $\mathcal{D} = (\Lambda_i)_{i=1,\dots,\mu}$ is a basis of $\mathscr{D}_{\mathbf{0}}$ and the normal form of any $g(\boldsymbol{x}) \in R$ with respect to the monomial basis $\mathcal{B} = (\boldsymbol{x}^{\boldsymbol{\beta}_i})_{i=1,\dots,\mu}$ is*

$$NF(g) = \sum_{i=1}^{\mu} \Lambda_i^{\mathbf{0}}[g]\, \boldsymbol{x}^{\boldsymbol{\beta}_i}. \tag{3.6}$$

Proof. First note that the elements of $\mathcal{D}$ are linearly independent, i.e. they form a basis. Now, by construction, $\sum_{i=1}^{\mu} \Lambda_i^{\mathbf{0}}[\boldsymbol{x}^{\boldsymbol{\alpha}}]\boldsymbol{x}^{\boldsymbol{\beta}_i} = \mathrm{NF}(\boldsymbol{x}^{\boldsymbol{\alpha}})$ for all $\boldsymbol{x}^{\boldsymbol{\alpha}} \notin \mathcal{Q}_{\mathbf{0}}$, e.g. $\mathrm{NF}(\boldsymbol{x}^{\boldsymbol{\beta}_i}) = \boldsymbol{x}^{\boldsymbol{\beta}_i}$. Also, for $\boldsymbol{x}^{\boldsymbol{\alpha}} \in \mathcal{Q}_{\mathbf{0}}$, $\forall i$, $\Lambda_i^{\mathbf{0}}(\boldsymbol{x}^{\boldsymbol{\alpha}}) = 0$, since $\boldsymbol{\alpha} \notin \operatorname{supp} \mathcal{D}$. Thus the elements of $\mathcal{D}$ compute $\mathrm{NF}(\cdot)$ on all monomials of R, and (3.6) follows by linearity. We deduce that $\mathcal{D}$ generates the dual, as in Def. 3.1. □

Computing the normal form of the border monomials of $\mathcal{B}$ via (3.6) also yields the border basis relations and the operators of multiplication in the quotient $R/\mathcal{Q}_{\mathbf{0}}$ (see e.g. [35, 102] for more properties).

If a graded monomial ordering is fixed and $\mathcal{B} = (\boldsymbol{x}^{\boldsymbol{\beta}_i})_{i=1,..,\mu}$ is the corresponding monomial basis of $R/\mathcal{Q}_{\mathbf{0}}$, then $\boldsymbol{d}^{\boldsymbol{\beta}_i}$ is the leading term of (3.5) w.r.t. the opposite ordering [66, Th. 3.1].

Conversely, if we are given a basis $\mathcal{D}$ of $\mathscr{D}_{\mathbf{0}}$ whose coefficient matrix in the dual monomials basis $(\boldsymbol{d}^{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \notin \mathcal{Q}_{\mathbf{0}}}$ is $D \in \mathbb{R}^{\mu \times s}$, we can compute a basis of $R/\mathcal{Q}_{\mathbf{0}}$ by choosing μ independent columns of D, say those indexed by $\boldsymbol{d}^{\boldsymbol{\beta}_i}$, $i = 1, \dots, \mu$. If $G \in \mathbb{R}^{\mu \times \mu}$ is the (invertible) matrix formed by these columns, then $D' := G^{-1}D$, is

$$D' = \begin{array}{c} \\ \Lambda'_1 \\ \vdots \\ \Lambda'_\mu \end{array} \begin{array}{c} \begin{array}{cccccc} \boldsymbol{\beta}_1 & \cdots & \boldsymbol{\beta}_\mu & \boldsymbol{\gamma}_1 & \cdots & \boldsymbol{\gamma}_{s-\mu} \end{array} \\ \begin{bmatrix} 1 & & 0 & \lambda_{1,1} & \cdots & \lambda_{1,s-\mu} \\ & \ddots & & \vdots & & \vdots \\ 0 & & 1 & \lambda_{\mu,1} & \cdots & \lambda_{\mu,s-\mu} \end{bmatrix} \end{array}, \tag{3.7}$$

i.e. a basis of the form (3.5). Note that an arbitrary basis of $\mathscr{D}$ does not have the above diagonal form, nor does it directly provide a basis for $R/\mathcal{Q}_{\mathbf{0}}$.

For $t \in \mathbb{N}$, $\mathscr{D}_t$ denotes the vector space of polynomials of $\mathscr{D}$ of degree $\leq t$. The Hilbert function $h : \mathbb{N} \to \mathbb{N}$ is defined by $h(t) = \dim(\mathscr{D}_t)$, $t \geq 0$, hence $h(0) = 1$ and $h(t) = \dim \mathscr{D}$ for $t \geq N$. The integer $h(1) - 1 =$ corank $J_{\boldsymbol{f}}$ is known as the *breadth* of $\mathscr{D}$.

3.3 Computing local ring structure

The computation of a local basis, given a system and a point, is done essentially by matrix-kernel computations, and consequently it can be carried out numerically, even when the point or even the system is inexact. Throughout the section we suppose $\boldsymbol{f} \in R^m$ and $\boldsymbol{\zeta} \in \mathbb{R}^n$ with $\boldsymbol{f}(\boldsymbol{\zeta}) = 0$.

Several matrix constructions have been proposed, that use different conditions to identify the dual space as a null-space. They are based on the *stability property* of the dual basis:

$$\forall\, \Lambda \in \mathscr{D}_t, \quad \frac{d}{d\partial_i}\Lambda \in \mathscr{D}_{t-1}\,, \quad i = 1, \ldots, n. \tag{3.8}$$

We list existing algorithms that compute dual-space bases:

- ◇ As pointed out in (3.3), an equivalent form of (3.8) is: $\forall \Lambda \in \mathscr{D}_t, \Lambda[x_i f_j] = \mathbf{0}$, $\forall i, j = 1, \ldots, n$. Macaulay's method [69] uses it to derive the algorithm that is outlined in Sect. 3.3.1.

- ◇ In [75] they exploit (3.8) by forming the matrix D_i of the map $\dfrac{d}{d\partial_i} : \mathbb{R}[\boldsymbol{\partial}]_t \to \mathbb{R}[\boldsymbol{\partial}]_{t-1}$ for all $i = 1, \ldots, n$ and some triangular decomposition of the differential polynomials in terms of differential variables. This approach was used in [99] to reduce the row dimension of Macaulay's matrix, but not the column dimension. The closedness condition is also used in [116] to identify a superset of supp $\mathscr{D}_{t+1}$.

- ◇ The *integration method* in [79] "integrates" elements of a basis of $\mathscr{D}_t$, and obtains *a priori* knowledge of the form of elements in degree $t + 1$ (Sect. 3.3.2).

All methods are incremental, in the sense that they start by setting $\mathcal{D}_0 = (\mathbf{1})$ and continue by computing $\mathcal{D}_i$, $i = 1, \ldots, N, N + 1$. When $\#\mathcal{D}_N = \#\mathcal{D}_{N+1}$ then $\mathcal{D}_N$ is a basis of $\mathscr{D}$, and N is the nilindex of $\mathcal{Q}$.

We shall review two of these approaches to compute a basis for $\mathscr{D}$, and then describe an improvement, that allows simultaneous computation of a monomial basis of the quotient ring while avoiding redundant computations.

3.3.1 Macaulay's dialytic matrices

This matrix construction is presented in [69, Ch. 4], a modern introduction is contained in [26], together with an implementation of the method in ApaTools*.

The idea behind the algorithm is the following: An element of $\mathscr{D}$ is of the form $\Lambda(\boldsymbol{d}) = \sum_{|\boldsymbol{\alpha}|\leq N} \lambda_{\boldsymbol{\alpha}} \boldsymbol{d}^{\boldsymbol{\alpha}}$ under the condition: $\Lambda^{\mathbf{0}}$ evaluates to 0 at any $g \in \langle \boldsymbol{f} \rangle$, i.e. $\Lambda^{\mathbf{0}}(g) = \Lambda^{\mathbf{0}}(\sum g_i f_i) = 0 \iff \Lambda^{\mathbf{0}}(x^{\boldsymbol{\beta}} f_i) = 0$ for all monomials $x^{\boldsymbol{\beta}}$. If we apply this condition recursively for $|\boldsymbol{\alpha}| \leq N$ we get a vector of coefficients $(\lambda_{\boldsymbol{\alpha}})_{|\boldsymbol{\alpha}|\leq N}$ in the (right) kernel of the matrix with rows indexed by constraints $\Lambda^{\mathbf{0}}[\boldsymbol{x}^{\boldsymbol{\beta}} \boldsymbol{f}_i] = 0$, $|\boldsymbol{\beta}| \leq N - 1$.

Note that the only requirement is to be able to perform derivation of the input equations and evaluation at $\boldsymbol{\zeta} = \mathbf{0}$.

Example 3.5. *Let $f_1 = x_1 - x_2 + x_1^2$, $f_2 = x_1 - x_2 + x_2^2$. We also refer the reader to [26, Ex. 2] for a detailed demonstration on this instance. The matrices in order 1 and 2 are:*

$$\begin{array}{c} \\ f_1 \\ f_2 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 1 & d_1 & d_2 \end{array} \\ \begin{bmatrix} 0 & 1 & -1 \\ 0 & 1 & -1 \end{bmatrix} \end{array}, \quad \begin{array}{c} \\ f_1 \\ f_2 \\ x_1 f_1 \\ x_1 f_2 \\ x_2 f_1 \\ x_2 f_2 \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} 1 & d_1 & d_2 & d_1^2 & d_1 d_2 & d_2^2 \end{array} \\ \begin{bmatrix} 0 & 1 & -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \end{array}.$$

The kernel of the left matrix gives $\mathcal{D}_1 = (1, d_1 + d_2)$. Expanding up to order two, we get the matrix on the right, and $\mathcal{D}_2 = (1, d_1 + d_2, -d_1 + d_1^2 + d_1 d_2 + d_2^2)$. If we expand up to depth 3 we get the same null-space, thus $\mathcal{D} = \mathcal{D}_2$.

3.3.2 Integration method

This method is presented in [79]. It is an evolution of Macaulay's method, since the matrices are not indexed by all differentials, but just by elements based on knowledge of the previous step. This performs a computation adapted to the given input and results in smaller matrices.

For $\Lambda \in \mathbb{R}[\boldsymbol{\partial}]$, we denote by $\int_k \Lambda$ the element $\Phi \in \mathbb{R}[\boldsymbol{\partial}]$ with the property $\frac{d}{d\partial_k}\Phi(\boldsymbol{\partial}) = \Lambda(\boldsymbol{\partial})$ and with no constant term w.r.t. ∂_k.

Theorem 3.6 ([79, Th. 15])**.** *Let $\langle \Lambda_1, \Lambda_2, \ldots, \Lambda_s \rangle$ be a basis of $\mathscr{D}_{t-1}$, that is, the subspace of $\mathscr{D}$ of elements of order at most $t-1$. An element $\Lambda \in \mathbb{R}[\boldsymbol{\partial}]$ with no*

*http://www.neiu.edu/~zzeng/apatools.htm

constant term lies in $\mathscr{D}_t$ iff it is of the form:

$$\Lambda(\boldsymbol{\partial}) = \sum_{i=1}^{s} \sum_{k=1}^{n} \lambda_{ik} \int_k \Lambda_i(\partial_1, \ldots, \partial_k, 0, \ldots, 0), \tag{3.9}$$

for $\lambda_{ik} \in \mathbb{R}$, and the following two conditions hold:

(i) $\displaystyle\sum_{i=1}^{s} \lambda_{ik} \frac{d}{d\partial_l} \Lambda_i(\boldsymbol{\partial}) - \sum_{i=1}^{s} \lambda_{il} \frac{d}{d\partial_k} \Lambda_i(\boldsymbol{\partial}) = 0,$
for all $1 \leq k < l \leq n$.

(ii) $\Lambda^{\boldsymbol{\zeta}}[f_k] = 0$, *for* $k = 1, \ldots, m$.

Condition (i) is equivalent to $\frac{d}{d\partial_k}\Lambda \in \mathscr{D}_{t-1}$, for all k. Thus the two conditions express exactly the fact that $\mathscr{D}$ must be stable under derivation and its members must vanish on $\langle \boldsymbol{f} \rangle$.

This gives the following algorithm to compute the dual basis: Start with $\mathcal{D}_0 = \langle 1 \rangle$. Given a basis of $\mathscr{D}_{t-1}$ we generate the ns candidate elements $\int_k \Lambda_{i-1}(\partial_1, \ldots, \partial_k, 0, \ldots, 0)$. Conditions (i) and (ii) give a linear system with unknowns λ_{ik}. The columns of the corresponding matrix are indexed by the candidate elements. Then, the kernel of this matrix gives a basis of $\mathscr{D}_t$, which we use to generate new candidate elements. If for some t we compute a kernel of the same dimension as $\mathscr{D}_{t-1}$, then we have a basis of $\mathscr{D}$.

Example 3.7. *Consider the instance of Ex. 3.5. We have $f_1(\boldsymbol{\zeta}) = f_2(\boldsymbol{\zeta}) = 0$, thus we set $\mathcal{D}_0 = \{1\}$. Equation (3.9) gives $\Lambda = \lambda_1 d_1 + \lambda_2 d_2$. Condition (i) induces no constraints and (ii) yields the system*

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = 0 \tag{3.10}$$

where the columns are indexed by d_1, d_2. We get $\lambda_1 = \lambda_2 = 1$ from the kernel of this matrix, thus $\mathcal{D}_1 = \{1, d_1 + d_2\}$.

For the second step, we compute the elements of $\mathcal{D}_2$, that must be of the form $\Lambda = \lambda_1 d_1 + \lambda_2 d_2 + \lambda_3 d_1^2 + \lambda_4 (d_1 d_2 + d_2^2)$. Condition (i) yields $\lambda_3 - \lambda_4 = 0$, and together with (ii) we form the system

$$\begin{bmatrix} 0 & 0 & 1 & -1 \\ 1 & -1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_4 \end{bmatrix} = 0, \tag{3.11}$$

with columns indexed by $d_1, d_2, d_1^2, d_1 d_2 + d_2^2$. We get two vectors in the kernel, the first yielding again $d_1 + d_2$ and a second one for $\lambda_1 = -1, \lambda_2 = 0, \lambda_3 = \lambda_4 = 1$, so we deduce that $-d_1 + d_1^2 + d_1 d_2 + d_2^2$ is a new element of $\mathcal{D}_2$.

In the third step we have

$$\begin{aligned}\Lambda =&\lambda_1 d_1 + \lambda_2 d_2 + \lambda_3 d_1^2 + \lambda_4(d_1 d_2 + d_2^2) + \\ &\lambda_5(d_1^3 - d_1^2) + \lambda_6(d_2^3 + d_1 d_2^2 + d_1^2 d_2 - d_1 d_2),\end{aligned} \tag{3.12}$$

condition (i) leads to $\lambda_3 - \lambda_4 + \lambda_6 + (\lambda_5 - \lambda_6)(d_1 + d_2) = 0$*, and together with condition (ii) we arrive to*

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_6 \end{bmatrix} = 0, \tag{3.13}$$

i.e. a 4×6 *matrix with two kernel elements that are already in* $\mathcal{D}_2$*. We derive that* $\mathscr{D} = \langle \mathcal{D}_2 \rangle = \langle \mathcal{D}_3 \rangle$ *and the algorithm terminates.*

Note that for this example Macaulay's method ends with a matrix of size 12×10*, instead of* 4×6 *in this approach.*

3.3.3 Computing a primal-dual pair

In this section we provide a process that allows simultaneous computation of a basis pair $(\mathcal{D}, \mathcal{B})$ of $\mathscr{D}$ and $R/\mathcal{Q}$.

Computing a basis of $\mathscr{D}$ degree by degree involves duplicated computations. The successive spaces computed are $\mathscr{D}_1 \subset \cdots \subset \mathscr{D}_N = \mathscr{D}_{N+1}$. It is more efficient to produce only new elements $\Lambda \in \mathscr{D}_t$, independent in $\mathscr{D}_t/\mathscr{D}_{t-1}$, at step t.

Also, once a dual basis is computed, one has to transform it to the form (3.5), in order to identify a basis of $R/\mathcal{Q}$ as well. This transformation can be done *a posteriori*, by finding a sub-matrix of full rank and then performing Gauss-Jordan elimination over this sub-matrix, to reach matrix form (3.7).

We introduce a condition (iii) extending Th. 3.6, that addresses these two issues: It allows the computation of a total of μ independent elements throughout execution, and returns a "triangular" basis, e.g. a basis of $R/\mathcal{Q}$ is identified.

Lemma 3.8. *Let* $\mathcal{D}_{t-1} = (\Lambda_1, \ldots, \Lambda_k)$ *be a basis of* $\mathscr{D}_{t-1}$*, whose coefficient matrix is*

$$\begin{array}{c} \\ \Lambda_1 \\ \vdots \\ \Lambda_k \end{array} \begin{array}{c} \begin{array}{cccccc} \boldsymbol{\beta}_1 & \cdots & \boldsymbol{\beta}_k & \gamma_1 & \cdots & \gamma_{s-k} \end{array} \\ \begin{bmatrix} 1 & * & * & * & \cdots & * \\ 0 & \ddots & * & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & * & \cdots & * \end{bmatrix} \end{array}, \tag{3.14}$$

yielding the monomial basis $\mathcal{B}_{t-1} = (\boldsymbol{x}^{\boldsymbol{\beta}_i})_{i=1,\ldots,k}$*. Also, let* $\Lambda \in \mathbb{R}[\boldsymbol{\partial}]$ *be of the form* (3.9)*, satisfying (i-ii) of Th. 3.6.*
If we impose the additional condition:

(iii) $\Lambda^{\varsigma}[\boldsymbol{x}^{\boldsymbol{\beta}_i}] = 0$, $1 \leq i \leq k$,

then the kernel of the matrix implied by (i-iii) is isomorphic to $\mathscr{D}_t/\mathscr{D}_{t-1}$. *Consequently, it extends* $\mathcal{D}_{t-1}$ *to a basis of* $\mathscr{D}_t$.

Proof. Let S be the kernel of the matrix implied by (i-iii), and let $\Lambda \in \mathbb{R}[\boldsymbol{\partial}]$ be a non-zero functional in S. We have $\Lambda \in \mathscr{D}_t$ and $\Lambda^{\varsigma}[\boldsymbol{x}^{\boldsymbol{\beta}_i}] = 0$ for $i = 1, \ldots, k$.

First we show that $\Lambda \notin \mathscr{D}_{t-1}$. If $\Lambda \in \mathscr{D}_{t-1}$, then $\Lambda = \sum_{i=1}^{k} \lambda_i \Lambda_i$. Take for i_0 the minimal i such that $\lambda_i \neq 0$. Then $\Lambda^{\varsigma}[\boldsymbol{x}^{\boldsymbol{\beta}_{i_0}}] = \lambda_{i_0}$, which contradicts condition (iii). Therefore, $S \cap \mathscr{D}_{t-1} = \{0\}$, and S can be naturally embedded in $\mathscr{D}_t/\mathscr{D}_{t-1}$, i.e. $\dim S \leq \dim \mathscr{D}_t - \dim \mathscr{D}_{t-1}$.

It remains to show that $\dim S$ is exactly $\dim \mathscr{D}_t - \dim \mathscr{D}_{t-1}$. This is true, since with condition (iii) we added $k = \dim \mathscr{D}_{t-1}$ equations, thus we excluded from the initial kernel (equal to $\mathscr{D}_t$) of (i-ii) a subspace of dimension at most $k = \dim \mathscr{D}_{t-1}$, so that $\dim S \geq \dim \mathscr{D}_t - \dim \mathscr{D}_{t-1}$.

We deduce that $S \cong \mathscr{D}_t/\mathscr{D}_{t-1}$, thus a basis of S extends $\mathcal{D}_{t-1}$ to a basis of $\mathscr{D}_t$. □

The above condition is easy to realize; it is equivalent to $\forall i$, $\boldsymbol{d}^{\boldsymbol{\beta}_i} \notin \operatorname{supp} \Lambda$, which implies adding a row (linear constraint) for every i.

If we choose the elements of $\mathcal{B}$ with an opposite to total degree ordering, this constraint becomes $\lambda_{ik} = 0$ for some i, k, thus we rather remove the column corresponding to λ_{ik} instead of adding a row. Hence this lemma allows to shrink the kernel (but also the dimension) of the matrix and compute only new dual elements, which are reduced modulo the previous basis.

Let us explore our running example, to demonstrate the essence of this improvement.

Example 3.9. *We re-run Ex. 3.7 using Lem. 3.8.*

In the initialization step $\mathcal{D}_0 = (1)$ *is already in triangular form with respect to* $\mathcal{B}_0 = \{1\}$. *For the first step, we demand* $\Lambda[1] = 0$, *thus the matrix is the same as* (3.10), *yielding* $\mathcal{D}_1 = (1, d_1 + d_2)$. *We extend* $\mathcal{B}_1 = \{1, x_2\}$, *so that* $\mathcal{D}_1$ *is triangular with respect to* $\mathcal{B}_1$.

In the second step we remove from (3.11) *the second column, hence we are left with a the* 3×3 *system in variables* $\lambda_1, \lambda_3, \lambda_4$,

$$\begin{bmatrix} 0 & 1 & -1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = 0,$$

yielding a single kernel vector $-d_1 + d_1^2 + d_1 d_2 + d_2^2$. *We extend* $\mathcal{B}_1$ *by adding monomial* x_1*:* $\mathcal{B}_2 = \{1, x_2, x_1\}$.

For the final step, we search an element of the form (3.12) *with* $\Lambda[x_1] = \Lambda[x_2] = 0$*, and together with (i-ii) we get:*

$$\begin{bmatrix} 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_3 \\ \vdots \\ \lambda_6 \end{bmatrix} = 0.$$

We find an empty kernel, thus we recover the triangular basis $\mathcal{D} = \mathcal{D}_2$*, which can be diagonalized to reach the form:*

$$\begin{array}{c} \\ \Lambda_1 \\ \Lambda_2 \\ \Lambda_3 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & d_2 & d_1 & d_1^2 & d_1d_2 & d_2^2 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & -1 & -1 & -1 \end{bmatrix} \end{array}.$$

This diagonal basis is dual to the basis $\mathcal{B} = (1, x_2, x_1)$ *of the quotient ring and also provides a normal form algorithm (Lem. 3.4) w.r.t.* $\mathcal{B}$*. In the final step we generated a* 4×4 *matrix, whose size is smaller compared to all previous methods.*

This technique for computing $\mathcal{B}$ can be applied similarly to other matrix methods, e.g. Macaulay's dialytic method.

If $h(t) - h(t-1) > 1$, i.e. there is more than one element in step t, then the choice of monomials to add to $\mathcal{B}$ is obtained by extracting a non-zero maximal minor from the coefficient matrix in $(\boldsymbol{d}^{\boldsymbol{\alpha}})$. In practice, we will look first at the minimum monomials w.r.t. a fixed term ordering.

3.3.4 Approximate dual basis

In our deflation method, we assume that the multiple point is known approximately and we use implicitly Taylor's expansion of the polynomials at this approximate point to deduce the dual basis, applying the algorithm of the previous section. To handle safely the numerical problems which may occur, we utilize the following techniques:

- ◇ At each step, the solutions of linear system (3.9, i-iii) are computed via Singular Value Decomposition. Using a given threshold, we determine the numerical rank and an orthogonal basis of the solutions from the last singular values and the last columns of the right factor of the SVD.
- ◇ For the computation of the monomials which define the equations (3.8, iii) at the next step, we apply QR decomposition on the transpose of the basis to extract a non-zero maximal minor. The monomials indexing this minor are used to determine constraints (3.9, i-iii). A similar numerical technique is employed in [116], for Macaulay's method.

3.4 Deflation of a singular point

We consider a system of equations $\boldsymbol{f} = (f_1, \ldots, f_s)$, $f_k \in \mathbb{R}[\boldsymbol{x}]$, which has a multiple root at $\boldsymbol{x} = \boldsymbol{\zeta}$. Also, let $\mathcal{B} = (\boldsymbol{x}^{\boldsymbol{\beta}_1}, \ldots, \boldsymbol{x}^{\boldsymbol{\beta}_\mu})$ be a basis of $R/\mathcal{Q}_{\boldsymbol{\zeta}}$ and $\mathcal{D} = (\Lambda_1, \ldots, \Lambda_\mu)$ its dual counterpart, with $\Lambda_1 = \mathbf{1}$.

We introduce a new set of equations starting from $\boldsymbol{f}$, as follows: add for every f_k the polynomial $g_k = f_k + p_k$, $p_k = \sum_{i=1}^{\mu} \varepsilon_{i,k}(\boldsymbol{x} - \boldsymbol{\zeta})^{\boldsymbol{\beta}_i}$ where $\boldsymbol{\varepsilon}_k = (\varepsilon_{k,1}, \ldots, \varepsilon_{k,\mu})$ is a new vector of μ variables.

Consider the system

$$\mathcal{D}\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = \Big(\Lambda_1(\boldsymbol{\partial}_{\boldsymbol{x}})[\boldsymbol{g}], \ldots, \Lambda_\mu(\boldsymbol{\partial}_{\boldsymbol{x}})[\boldsymbol{g}]\Big).$$

where $\Lambda^{\boldsymbol{x}}[g_k] = \Lambda_i(\boldsymbol{d}_{\boldsymbol{x}})[g_k]$ is defined as in (3.2) with $\boldsymbol{\zeta}$ replaced by $\boldsymbol{x}$, i.e. we differentiate g_k but we do not evaluate at $\boldsymbol{\zeta}$. This is a system of μs equations, which we shall index $\mathcal{D}\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = (g_{1,1}, \ldots, g_{\mu,s})$. We have

$$g_{ik}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = \Lambda_i^{\boldsymbol{x}}[f_k + p_k] = \Lambda_i^{\boldsymbol{x}}[f_k] + \Lambda_i^{\boldsymbol{x}}[p_k] = \Lambda_i^{\boldsymbol{x}}[f_k] + p_{i,k}(\boldsymbol{x}, \boldsymbol{\varepsilon}).$$

Notice that $p_{i,k}(\boldsymbol{\zeta}, \boldsymbol{\varepsilon}) = \Lambda_i^{\boldsymbol{\zeta}}[p_k] = \varepsilon_{i,k}$ because $\mathcal{D} = (\Lambda_1, .., \Lambda_\mu)$ is dual to $\mathcal{B}$.

As the first basis element of $\mathcal{D}$ is $\mathbf{1}$ (the evaluation at the root), the first s equations are $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = 0$.

Note that this system is under-determined, since the number of variables is $\mu\, s + n$ and the number of equations is μs. We shall provide a systematic way to choose n variables and purge them (or better, set them equal to zero).

This way we arrive to a square system $\mathcal{D}\boldsymbol{g}(\boldsymbol{x}, \tilde{\boldsymbol{\varepsilon}})$ (we use $\tilde{\boldsymbol{\varepsilon}}$ for the remaining $\mu s - n$ variables) of size $\mu s \times \mu s$. We shall prove that this system vanishes on $(\boldsymbol{\zeta}, \mathbf{0})$ and that $J_{\mathcal{D}\boldsymbol{g}}(\boldsymbol{\zeta}, \mathbf{0}) \neq 0$.

By linearity of the Jacobian matrix we have

$$\begin{aligned} J_{\mathcal{D}\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) &= J_{\mathcal{D}\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) + J_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) \\ &= [\, J_{\mathcal{D}\boldsymbol{f}}(\boldsymbol{x}) \,|\, \mathbf{0} \,] + [\, J^{\boldsymbol{x}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) \,|\, J^{\boldsymbol{\varepsilon}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) \,], \end{aligned}$$

where $J^{\boldsymbol{x}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon})$ (resp. $J^{\boldsymbol{\varepsilon}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon})$) is the Jacobian matrix of $\mathcal{D}\boldsymbol{p}$ with respect to $\boldsymbol{x}$ (resp. $\boldsymbol{\varepsilon}$).

Lemma 3.10. *The Jacobian $J^{\boldsymbol{\varepsilon}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon})$ of the linear system $\mathcal{D}\boldsymbol{p} = (p_{1,1}, \ldots, p_{\mu,s})$ with $p_{i,k}(\boldsymbol{\varepsilon}_k) = \Lambda_i^{\boldsymbol{x}}[p_k](\boldsymbol{x}, \boldsymbol{\varepsilon}_k)$ evaluated at $(\boldsymbol{x}, \boldsymbol{\varepsilon}) = (\boldsymbol{\zeta}, \mathbf{0})$ is the identity matrix of dimension μs.*

Proof. First note that the system is block separated, i.e. every $p_{i,k}$ depends only on variables $\boldsymbol{\varepsilon}_k$ and not on all $\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_s)$. This shows that $J^{\boldsymbol{\varepsilon}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon})$ is block diagonal, $J^{\boldsymbol{\varepsilon}}_{\mathcal{D}\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = \mathrm{diag}(J_1, \ldots, J_\mu)$. Now we claim that these blocks

are all equal to the identity matrix. To see this, consider their entry $\frac{d}{d\varepsilon_{k,j}}p_{i,k}$ for $i,j=1,..,\mu$, which is

$$\frac{d}{d\varepsilon_{k,j}}\Lambda_i^{\boldsymbol{\zeta}}[p_k]=\Lambda_i^{\boldsymbol{\zeta}}[\frac{d}{d\varepsilon_{k,j}}p_k]=\Lambda_i^{\boldsymbol{\zeta}}[\boldsymbol{x}^{\boldsymbol{\beta}_j}]=\begin{cases}1 & ,i=j\\ 0 & ,\text{otherwise}\end{cases},$$

since $\frac{d}{d\varepsilon_{k,j}}p_k=\frac{d}{d\varepsilon_{k,j}}(\boldsymbol{x}^{\boldsymbol{\beta}_j}\varepsilon_{k,j})=\boldsymbol{x}^{\boldsymbol{\beta}_j}$. □

Lemma 3.11. *The $\mu s\times n$ Jacobian matrix $J_{\mathcal{D}\boldsymbol{f}}(\boldsymbol{x})$ of the system $\mathcal{D}\boldsymbol{f}(\boldsymbol{x})=(f_1,\ldots,f_{\mu n})$ is of full rank n at $\boldsymbol{x}=\boldsymbol{\zeta}$.*

Proof. Suppose that the matrix is rank-deficient. Then there is a non-trivial vector in its kernel,

$$J_{\mathcal{D}\boldsymbol{f}}(\boldsymbol{\zeta})\cdot\boldsymbol{v}=\mathbf{0}.$$

The entries of $\boldsymbol{v}$ are indexed by ∂_i. This implies that a non-zero differential $\Delta=v_1\partial_1+\cdots+v_n\partial_n$ of order one satisfies the following relations: $(\Delta\Lambda_i)^{\boldsymbol{\zeta}}[f_j]=0, i=1,\ldots,\mu, j=1,\ldots,s$. By the standard derivation rules, we have

$$\frac{d}{d\partial_k}(\Delta\Lambda_i)=v_k\Lambda_i+\Delta\frac{d}{d\partial_k}\Lambda_i,$$

for $i=1,\ldots,\mu,,k=1,\ldots,n$. Since $\mathscr{D}$ is stable under derivation, $\frac{d}{d\partial_k}\Lambda_i\in\mathscr{D}$. We deduce that the vector space spanned by $\langle\mathscr{D},\Delta\mathscr{D}\rangle$ is stable under derivation and vanishes on $\boldsymbol{f}$ at $\boldsymbol{\zeta}$. By Proposition 3.2, we deduce that $\Delta\mathscr{D}\subset\mathscr{D}$. This is a contradiction, since Δ is of degree 1 and the elements in $\mathscr{D}$ are of degree $\leq N$. □

The columns of $J_{\mathcal{D}\boldsymbol{g}}(\boldsymbol{x},\boldsymbol{\varepsilon})$ are indexed by the variables $(\boldsymbol{x},\boldsymbol{\varepsilon})$, while the rows are indexed by the polynomials g_{ik}. We construct the following systems:

(a) Let $\mathcal{D}\boldsymbol{f}^I$ be a subsystem of $\mathcal{D}\boldsymbol{f}$ s.t. the corresponding n rows of $J_{\mathcal{D}\boldsymbol{f}}(\boldsymbol{\zeta})$ are linearly independent (Lem. 3.11). We denote by $I=\{(i_1,k_1),\ldots,(i_n,k_n)\}$ their indices.

(b) Let $\mathcal{D}\tilde{\boldsymbol{g}}(\boldsymbol{x},\tilde{\boldsymbol{\varepsilon}})$ be the square system formed by removing the variables $\varepsilon_{k_1,i_1},\ldots,\varepsilon_{k_n,i_n}$ from $\mathcal{D}\boldsymbol{g}(\boldsymbol{x},\boldsymbol{\varepsilon})$. Therefore the Jacobian $J_{\mathcal{D}\tilde{\boldsymbol{g}}}(\boldsymbol{x},\tilde{\boldsymbol{\varepsilon}})$ derives from $J_{\mathcal{D}\boldsymbol{g}}(\boldsymbol{x},\boldsymbol{\varepsilon})$, after purging the columns indexed by $\varepsilon_{k_1,i_1},\ldots,\varepsilon_{k_n,i_n}$ and it's $(i_j,k_j)-$th row becomes $[\nabla(\Lambda_{i_j}^{\boldsymbol{x}}\tilde{g}_{i_j,k_j})^T|\,\mathbf{0}\,]$.

Theorem 3.12 (Deflation Theorem 1)**.** *Let $\boldsymbol{f}(\boldsymbol{x})$ be a $n-$variate polynomial system with an $\mu-$fold isolated zero at $\boldsymbol{x}=\boldsymbol{\zeta}$. Then the $n\times n$ system $\mathcal{D}\boldsymbol{f}^I(\boldsymbol{x})=0$, defined in (a), has a simple root at $\boldsymbol{x}=\boldsymbol{\zeta}$.*

Proof. By construction, $\boldsymbol{\zeta}$ is a solution of $\mathcal{D}\boldsymbol{f}^I(\boldsymbol{x})=0$. Moreover, the indices I are chosen such that $\det J_{\mathcal{D}\boldsymbol{f}^I}(\boldsymbol{\zeta})\neq 0$. This shows that $\boldsymbol{\zeta}$ is a simple (thus isolated) root of the system $\mathcal{D}\boldsymbol{f}^I(\boldsymbol{x})=0$. □

Example 3.13. *In our running example, we expand the rectangular Jacobian matrix of* 6 *polynomials in* (x_1, x_2). *Choosing the rows corresponding to* f_1 *and* $(d_1 - d_2^2 - d_1 d_2 - d_1^2)[f_1]$, *we find a non-singular minor, hence the resulting system* $(f_1, 2x_1)$ *has a regular root at* $\boldsymbol{\zeta} = (0, 0)$.

The deflated system $\mathcal{D}\boldsymbol{f}^I(\boldsymbol{x}) = 0$ is a square system in n variables. Contrarily to the deflation approach in [26, 65], we do not introduce new variables and one step of deflation is sufficient. The trade-off is that here we assume that exact dual elements are pointed at by indices I, so as to be able to compute the original multiple root with high accuracy.

On the other hand, when the coefficients are machine numbers, an exact multiple root is unlikely to exist. In the following theorem, we introduce new variables that will allow us later to derive an approximate deflation method.

Theorem 3.14 (Deflation Theorem 2)**.** *Let* $\boldsymbol{f}(\boldsymbol{x})$ *be a* $n-$*variate polynomial system with a* $\mu-$*fold isolated root at* $\boldsymbol{x} = \boldsymbol{\zeta}$. *The* square *system* $\mathcal{D}\tilde{\boldsymbol{g}}(\boldsymbol{x}, \tilde{\varepsilon}) = 0$, *as defined in (b), has a regular isolated root at* $(\boldsymbol{x}, \tilde{\varepsilon}) = (\boldsymbol{\zeta}, \mathbf{0})$.

Proof. Computing approximate dual basis at $\boldsymbol{\zeta}$ we get

$$\mathcal{D}\tilde{\boldsymbol{g}}(\boldsymbol{\zeta}, \mathbf{0}) = (\Lambda_1^{\boldsymbol{\zeta}}[\boldsymbol{f}], \ldots, \Lambda_\mu^{\boldsymbol{\zeta}}[\boldsymbol{f}]) = 0.$$

Moreover, by construction of $\mathcal{D}\tilde{\boldsymbol{g}}$ we get, up to a row permutation, the following structure in the Jacobian determinant:

$$\pm\det J_{\mathcal{D}\tilde{\boldsymbol{g}}}(\boldsymbol{\zeta}, \mathbf{0}) = \det \begin{vmatrix} J_1 & 0 \\ J_2 & I \end{vmatrix} = \det J_1 \neq 0,$$

where $J_1 = J_{\mathcal{D}\boldsymbol{f}^I}(\boldsymbol{\zeta})$. This shows that $(\boldsymbol{\zeta}, \mathbf{0})$ is regular and thus isolated point of the algebraic variety defined by $\mathcal{D}\tilde{\boldsymbol{g}}(\boldsymbol{x}, \tilde{\varepsilon}) = 0$. □

Nevertheless, this deflation does differ from the deflation strategy in [26, 65]. There, new variables are added that correspond to coefficients of differential elements, thus introducing a perturbation in the dual basis.. This is suitable for exact equations, but, in case of perturbed data, the equations do not actually define a true singular point. In our method, we perturb the equations, keeping a fixed structure of a multiple root. Consequently, the certification of a root concerns a nearby system, within controlled error bounds, that attains a true multiple point, as it shall be described in the next section.

We mention that it would also be possible to use the equations (3.9, i-iii) to construct a deflated system on the differentials and to perturb the approximate dual structure.

3.5 Verifying approximate singular points

In real-life CAD applications, it is common to work with approximate inputs. Also, there is the need to (numerically) decide if an (approximate) system possesses a single (real) root in a given domain, notably for use in subdivision-based algorithms, e.g. [72, 80].

In the regular case, Smale's $\alpha-$theory, extending Newton's method, can be used to answer this problem, also partially extended to singular cases in [47], using zero clustering. Another option is the following theorem, also based on Newton theory. In our implementation we use this latter approach, since it is suitable for inexact data and suits best with the perturbation which is applied. In particular, it coincides with the numerical scheme of [90] in the univariate case.

The certification test is based on interval analysis techniques. We refer to the following theorem due to Krawczyk, also Rump:

Theorem 3.15 ([56, 60, 90]). *Let $\boldsymbol{f} \in R^n$ be a polynomial system and $\boldsymbol{\zeta}^* \in \mathbb{R}^n$ a real point. Given an interval domain $Z \in \mathbb{IR}^n$ containing $\boldsymbol{\zeta}^* \in \mathbb{R}^n$, and an interval matrix $M \in \mathbb{IR}^{n\times n}$ whose $i-$th column M_i satisfies $\nabla f_i(Z) \subseteq M_i$ for $i = 1 \ldots, n$, then the following holds:*

If the interval domain

$$V_{\boldsymbol{f}}(Z, \boldsymbol{\zeta}^*) := -J_{\boldsymbol{f}}(\boldsymbol{\zeta}^*)^{-1}\boldsymbol{f}(\boldsymbol{\zeta}^*) + (I - J_{\boldsymbol{f}}(\boldsymbol{\zeta}^*)^{-1}M)Z \tag{3.15}$$

is contained in the interior of Z, then there is a unique $\boldsymbol{\zeta} \in Z$ with $\boldsymbol{f}(\boldsymbol{\zeta}) = \mathbf{0}$ and the Jacobian $J_{\boldsymbol{f}}(\boldsymbol{\zeta}) \in M$ is non-singular.

This theorem is applied to the system of 3.14, using an (approximate) structure $\mathcal{D}$. The resulting range of the $\boldsymbol{\varepsilon}-$parameters encloses a system that attains a single multiple root of that structure. Hence the domain for $\boldsymbol{\varepsilon}-$variables reflects the distance of the approximate system from an exact system with local structure $\mathcal{D}$, see Ex. 3.18.

3.6 Geometry around a singularity

As a final step in analyzing isolated singularities, we show how the local basis can be used to compute the topological degree around the singular point. If the latter is a singular point (e.g. self-intersection point, cusp, isolated point etc), of a real algebraic curve, one can deduce the number of curve (half-)branches that touch the point. We will not be interested in the type of the singularity; methods exist that can identify the type of a given singularity, eg. the symbolic-numeric method for Puiseux expansions of [88] or the works based on genus computation [53, 54, 55].

3.6.1 Topological degree computation

Let $\boldsymbol{f}(\boldsymbol{x})$ be a square $n-$variate system with an $\mu-$fold isolated zero at $\boldsymbol{x} = \boldsymbol{\zeta}$.

The topological degree $\text{tdeg}_{\boldsymbol{\zeta}}(\boldsymbol{f})$ at $\boldsymbol{x} = \boldsymbol{\zeta}$ is the number of times that the (Gauss) map $G_{\boldsymbol{\zeta}} : S_{\boldsymbol{\zeta}}(\varepsilon) \to \mathbb{S}^{n-1}$, $G_{\boldsymbol{\zeta}}(\boldsymbol{x}) := \boldsymbol{f}(x)/\|\boldsymbol{f}(\boldsymbol{x})\|$, with domain on the ball $S_{\boldsymbol{\zeta}}(\varepsilon)$, wraps around the sphere $\mathbb{S}^{n-1}$. This integer remains invariant if we replace spheres by any other compact oriented manifold [33].

To a functional $\Lambda \in \mathbb{R}[\boldsymbol{\partial}]$, we associate the quadratic form

$$Q_\Lambda : R/\mathcal{Q} \times R/\mathcal{Q} \to \mathbb{R} \quad , \quad (\boldsymbol{x}^{\boldsymbol{\beta}_i}, \boldsymbol{x}^{\boldsymbol{\beta}_j}) \mapsto \Lambda[\boldsymbol{x}^{\boldsymbol{\beta}_i+\boldsymbol{\beta}_j}] \tag{3.16}$$

for $R/\mathcal{Q} = \langle \boldsymbol{x}^{\boldsymbol{\beta}_1}, \ldots, \boldsymbol{x}^{\boldsymbol{\beta}_\mu} \rangle$. The signature of this (symmetric and bi-linear) form is the sum of signs of the diagonal entries of any diagonal matrix representation of it.

Proposition 3.16 ([33, Th. 1.2])**.** *If Q_Φ, $\Phi \in \mathcal{D}$ is any bi-linear symmetric form such that $\Phi^{\boldsymbol{\zeta}}[\det J_{\boldsymbol{f}}(\boldsymbol{x})] > 0$, then*

$$tdeg_{\boldsymbol{\zeta}}(\boldsymbol{f}) = sgn(Q_\Phi). \tag{3.17}$$

This signature is independent of the bi-linear form used.

We can use this result to compute the topological degree at $\boldsymbol{x} = \boldsymbol{\zeta}$ using the dual structure at $\boldsymbol{\zeta}$. Since a basis $\mathcal{D}$ is available we set $\Phi = \pm\Lambda_i$, for some basis element that is not zero on $\det J_{\boldsymbol{f}}(\boldsymbol{x})$. Indeed, such an element can be retrieved among the basis elements, since $\det J_{\boldsymbol{f}} \notin \langle \boldsymbol{f} \rangle$, see [35, Ch. 0].

In practice it suffices to generate a random element of $\mathscr{D}$, compute its matrix representation $[\Phi(\boldsymbol{x}^{\boldsymbol{\beta}_i+\boldsymbol{\beta}_j})]_{ij}$, and then extract the signature of Q_Φ.

3.6.2 Branches around a singularity

In the context of computing with real algebraic curves, the identification of singular points is only the first step towards determining the local topology. As a second step, one needs to calculate the number of half-branches attached to the singular point $\boldsymbol{\zeta}$, hereafter denoted $\text{Br}(\boldsymbol{f}, \boldsymbol{\zeta})$. This information is encoded in the topological degree.

An implicit curve in $n-$space is given by all points satisfying $\boldsymbol{f}(\boldsymbol{x}) = 0$, $\boldsymbol{f} = (f_1, \ldots, f_{n-1})$. Consider $p(\boldsymbol{x}) = (x_1 - \zeta_1)^2 + \cdots + (x_n - \zeta_n)^2$, and $g(\boldsymbol{x}) = \det J_{(\boldsymbol{f},p)}(\boldsymbol{x})$. Then ([100] and references therein):

$$\text{Br}(\boldsymbol{f}, \boldsymbol{\zeta}) = 2\,\text{tdeg}_{\boldsymbol{\zeta}}(\boldsymbol{f}, g). \tag{3.18}$$

This implies an algorithm for $\text{Br}(\boldsymbol{f}, \boldsymbol{\zeta})$. First compute the primal-dual structure of $(\boldsymbol{f}, g)$ at $\boldsymbol{\zeta}$ and then use Prop. 3.16 to get $\text{tdeg}_{\boldsymbol{\zeta}}(\boldsymbol{f}, g)$, see Ex. 3.19.

3.7 Experimentation

Our method is developed in MAPLE. It uses our modified integration technique to compute (approximate) dual basis and derive the augmented system of Th. 3.14. Then Rump's method is used to verify the root. Macaulay's method is also implemented for testing purposes.

Example 3.17. *Let, as in [63, 66],* $f_1 = 2x_1 + 2x_1^2 + 2x_2 + 2x_2^2 + x_3^2 - 1$, $f_2 = (x_1 + x_2 - x_3 - 1)^3 - x_1^3$, *and* $f_3 = 2x_1^3 + 2x_2^2 + 10x_3 + 5x_3^2 + 5)^3 - 1000x_1^5$.

Point $(0, 0, -1)$ *occurs with multiplicity equal to* 18, *in depth* 7. *The final matrix size with our method is* 206×45, *while Macaulay's method ends with a* 360×165 *matrix.*

If the objective is to deflate as efficiently as possible, then one can go step by step: First compute a basis of $\mathscr{D}_1$ *and stop the process. We get the evaluation* $\mathbf{1}$ *and* 2 *first order functionals, which we apply to* f_1. *We arrive to* $(\mathbf{1}[f_1], (d_2 - d_1)[f_1], (d_1 + d_3)[f_1]) = (f_1, -4x_1 + 4x_2, 2 + 4x_1 + 2x_3)$ *and we check that the Jacobian determinant is* 64, *thus we have a deflated system only with a partial local structure.*

Example 3.18. *Consider the equations ([26, DZ3]):* $f_1 = 14x_1 + 33x_2 - 3\sqrt{5}(x_1^2 + 4x_1x_2 + 4x_2^2 + 2) + \sqrt{7} + x_1^3 + 6x_1^2x_2 + 12x_1x_2^2 + 8x_2^3$, $f_2 = 41x_1 - 18x_2 - \sqrt{5} + 8x_1^3 - 12x_1^2x_2 + 6x_1x_2^2 - x_2^3 + 3\sqrt{7}(4x_1x_2 - 4x_1^2 - x_2^2 - 2)$ *and take an approximate system* $\tilde{\boldsymbol{f}}$ *with those coefficients rounded to* 6 *digits. A* $5-$*fold zero of* $\boldsymbol{f}$ *rounded to* 6 *digits is* $\boldsymbol{\zeta}^* = (1.50551, .365278)$.

Starting with the approximate system and with a tolerance of .001, *we compute the basis* $\mathcal{D} = (1, d_1 + .33d_2, d_1^2 + .33d_1d_2 + .11d_2^2, d_1^3 + .33d_1^2d_2 + .11d_1d_2^2 + .03d_2^3 - 1.54d_2, d_1^4 + .33d_1^3d_2 + .11d_1^2d_2^2 + .03d_1d_2^3 + .01d_2^4 - 1.54d_1d_2 - 1.03d_2^2)$ *having* 4 *correct digits, w.r.t. the initial exact system, and the primal counterpart* $\mathcal{B} = (1, x_1, x_1^2, x_1^3, x_1^4)$.

We form the deflated system (b), with $I = \{(3, 1), (5, 1)\}$, *i.e. the 3rd and 5th dual element on* f_1 *have non-null Jacobian. By adding 8 new variables, the system is perturbed as:* $g_{1,1} = \tilde{f}_1 + \varepsilon_{1,1} + \varepsilon_{1,2}(x_1 - \zeta_1^*) + \varepsilon_{1,4}(x_1 - \zeta_1^*)^3$, $g_{2,1} = \tilde{f}_2 + \sum_{i=1}^{5} \varepsilon_{2,i}(x_1 - \zeta_1^*)^{i+1}$ *and their derivation w.r.t.* $\mathcal{D}$.

We consider a box Z *with center* $= \boldsymbol{\zeta}^*$ *and length* $= .004$ *at each side. Also, we allow a range* $E = [-.004, .004]^8$ *for the variables* $\tilde{\boldsymbol{\varepsilon}}$. *Applying 3.15 we get a verified inclusion* $V_{\boldsymbol{g}}(Z \times E, (\boldsymbol{\zeta}^*, \mathbf{0}))$ *inside* $Z \times E$ *and we deduce that a unique specialization* $\tilde{\boldsymbol{\varepsilon}} \in E$ *"fits" the approximate system* $\tilde{\mathbf{f}}$ *to the multiplicity structure* $\mathcal{D}$.

Indeed, one iteration of Newton's method on $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\varepsilon})$ *gives* $\boldsymbol{\zeta} = (1.505535473, .365266196)$ *and corresponding values for* $\boldsymbol{\varepsilon}_0 \in E$, *such that* $\boldsymbol{\zeta}$ *is a* $9-$*digit approximation of the multiple root of the perturbed system* $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\varepsilon}_0)$.

Example 3.19. *Consider the implicit curve $f(x,y)=0$, in $xy-$plane, with*

$$f(x,y)=x^4+2x^2y^2+y^4+3x^2y-y^3,$$

that looks like this  . *We search for the number of half-branches touching $\boldsymbol{\zeta}=(0,0)$.*

This point is of multiplicity 4, as the dual basis we get for

$$f(x,y)=\frac{d}{dx}f(x,y)=\frac{d}{dy}f(x,y)=0$$

is $(1,d_x,d_y,d_x^2+d_y^2)$, yet this provides no information for the number of branches around the point $\boldsymbol{\zeta}$.

We compute

$$g(x,y)=J_{(f,x^2+y^2)}=18xy^2-6x^3,$$

and then the multiplicity structure of (f,g) at $\boldsymbol{\zeta}$, and we arrive to

$$\begin{aligned}\mathcal{D}=&(1,\ d_y,\ d_x,\ d_y^2,\ d_xd_y,\ d_x^2,\ \underline{d_y^3}+\frac{3}{8}\,d_y^4+\frac{1}{8}\,d_x^2d_y^2+\frac{3}{8}\,d_x^4,\\ &\underline{d_xd_y^2}+3\ d_x^3,\ \underline{d_x^2d_y}-\frac{9}{8}\,d_y^4-\frac{3}{8}\,d_x^2d_y^2-\frac{9}{8}\,d_x^4),\end{aligned}$$

and the monomial basis

$$\mathcal{B}=(1,y,x,y^2,xy,x^2,y^3,xy^2,x^2y).$$

Among the 9 elements of the dual basis, we find

$$\Phi=d_y^3+\frac{3}{8}d_y^4+\frac{1}{8}d_x^2d_y^2+\frac{3}{8}d_x^4,$$

having value $\Phi^{\mathbf{0}}[\det J_{(f,g)}(\boldsymbol{x})]=54>0$ on the Jacobian determinant.

Now a representation of Q_Φ (3.16) can be computed, by applying $\Phi^{\mathbf{0}}$ to $\boldsymbol{x}^{\boldsymbol{\beta}_i+\boldsymbol{\beta}_j}$ to get the $ij-$th entry of the matrix:

$$Q_\Phi=\begin{bmatrix}0&0&0&0&0&0&1&0&0\\0&0&0&1&0&0&3/8&0&1/8\\0&0&0&0&0&0&0&1/8&0\\0&1&0&3/8&0&1/8&0&0&0\\0&0&0&0&1/8&0&0&0&0\\0&0&0&1/8&0&3/8&0&0&0\\1&3/8&0&0&0&0&0&0&0\\0&0&1/8&0&0&0&0&0&0\\0&1/8&0&0&0&0&0&0&0\end{bmatrix}.$$

With a QR iteration, we find 6 positive and 3 negative eigenvalues of this matrix, hence we compute

$$tdeg_{\zeta}(f,g) = sgn\, Q_{\Phi} = 6 - 3 = 3,$$

i.e. there are 6 *half-branches of the curve around* $(0,0)$.

In Table 3.1 we run dual basis computation on the benchmark set of [26] in MAPLE. Multiplicity, matrix sizes at termination step and computation time is reported. One sees that there is at least an order of gain in the running time.

System	μ/n	**primal-dual**		**integration**		**Macaulay**	
cmbs1	11/3	27×23	.18s	27×33	.95s	105×56	1.55s
cmbs2	8/3	21×17	.08s	21×24	.39s	60×35	.48s
mth191	4/3	10×9	.03s	10×12	.07s	30×20	.14s
decker2	4/2	5×5	.02s	5×8	.05s	20×15	.10s
Ojika2	2/3	6×5	.02s	6×6	.03s	12×10	.04s
Ojika3	4/3	12×9	.07s	12×12	.27s	60×35	.59s
KSS	16/5	155×65	8.59s	155×80	40.41s	630×252	70.03s
Capr.	4/4	22×13	.28s	22×16	.47s	60×35	2.34s
Cyclic-9	4/9	104×33	1.04s	104×36	5.47s	495×220	31.40s
DZ1	131/4	700×394	14m	700×524	26m	4004×1365	220m
DZ2	16/3	43×33	.68s	43×48	4.38s	360×165	25.72s
DZ3	5/2	6×6	.04s	6×10	.23s	30×21	.79s

Table 3.1: Benchmark systems from [3].

Verification Example. Let $f_1 = x_1^2x_2 - x_1x_2^2$, $f_2 = x_1 - x_2^2$. The verification method of [90] applies a linear perturbation to this system, but fails to certify the root $\boldsymbol{x} = (0,0)$.

We consider an approximate point $\boldsymbol{\zeta}^* = (.01,\ .002)$ and we compute the approximate multiplicity structure

$$\mathcal{D} = (\Lambda_1, \ldots, \Lambda_4) = (1.0, 1.0d_2, \underline{1.0d_1} + 1.0d_2^2, \underline{1.0d_1d_2} + 1.0d_2^3).$$

The augmented system $\boldsymbol{g}(\boldsymbol{x}) = (\Lambda_j[f_i]) = (f_1,\ 2.0x_1x_2 - 1.0x_2^2 - 1.0x_1,\ 2.0x_1 - 2.0x_2,\ 1.0x_1 - 1.0x_2^2,\ f_2,\ -2.0x_2,\ 0.,0.)$ has a Jacobian matrix:

$$J_{\boldsymbol{g}}(\boldsymbol{\zeta}^*)^T = \begin{bmatrix} .00 & .016 & -.99 & 2.0 & 1.0 & 0 & 0 & 0 \\ .00 & -.02 & .016 & -2.0 & -.004 & -2.0 & 0 & 0 \end{bmatrix}$$

with a non-zero minor at the third and forth row. Using this information, we apply the following perturbation to the original system:

$$g_1 = x_1^2 x_2 - x_1 x_2^2 + \varepsilon_{11} + \varepsilon_{12} x_2$$
$$g_5 = x_1 - x_2^2 + \varepsilon_{21} + \varepsilon_{22} x_2 + \varepsilon_{23} x_1 + \varepsilon_{24} x_1 x_2$$

Thus $\boldsymbol{g}(x_1, x_2, \varepsilon_{11}, \varepsilon_{12}, \varepsilon_{21}, \varepsilon_{22}, \varepsilon_{23}, \varepsilon_{24})$, computed as before, is a square system with additional equations:

$$\begin{aligned}
g_2 &= 1.0x_1^2 - 2.0x_1x_2 + 1.0\varepsilon_{12} \\
g_3 &= 2.0x_1x_2 - 1.0x_2^2 - 1.0x_1 \\
g_4 &= 2.0x_1 - 2.0x_2 \\
g_6 &= -2.0x_2 + 1.0\varepsilon_{22} + 1.0x_1\varepsilon_{24} \\
g_7 &= 1.0\varepsilon_{23} + 1.0x_2\varepsilon_{24} \\
g_8 &= 1.0\varepsilon_{24}
\end{aligned}$$

Now take the box $Z_1 = [-.03, .05] \times [-.04, .04] \times [-.01, .01]^6$. We apply Th. 3.15 on $\boldsymbol{g}$, i.e. we compute $V_{\boldsymbol{g}}(Z_1, \boldsymbol{\zeta}^*)$. For the variable ε_{21} the interval is $[-.015, .15] \not\subseteq (-.01, .01)$, therefore we don't get an answer.

We shrink a little Z_1 down to $Z_2 = [-.03, .05] \times [-.02, .02] \times [-.01, .01]^6$ and we apply again Th. 3.15, which results in

$$V_{\boldsymbol{g}}(Z_2, (\boldsymbol{\zeta}^*, \mathbf{0})) = \begin{bmatrix} [-.004, .004] \\ [-.004, .004] \\ [-.001, .001] \\ [-.007, .007] \\ [-.006, .006] \\ [-.009, .009] \\ [-.00045, .00035] \\ [.0, .0] \end{bmatrix} \subseteq \overset{\circ}{Z}_2,$$

thus we certify the multiple root of the original system inside $[-.03, .05] \times [-.02, .02]$. □

Example 3.20. *We start with an approximate system:* $f_1 = 1.071x_1 - 1.069x_2 + 1.018x_1^2$, $f_2 = 1.024x_1 - 1.016x_2 + 1.058x_2^2$ *and the domain:* $Z = [-.01, .03] \times [-.03, .01]$. *The Jacobian at* $\boldsymbol{x} = (0, 0)$ *evaluates to* $.00652$*, hence it is close to singular.*

We set the tolerance equal to $.04$*, i.e. the size of the domain, and we consider the center of the box as our approximate point,* $\boldsymbol{\zeta}^* = (.01, -.01)$.

First we compute approximate multiplicity structure at $\boldsymbol{\zeta}^*$, $\mathcal{D} = (1, d_2 + 1.00016d_2^2 + .99542d_1d_2 + 1.03023d_1^2, d_1 - 1.00492d_2^2 - 1.00016d_1d_2 - 1.03514d_1^2)$

as well as $(1, x_2, x_1)$*, a monomial basis for the quotient. The latter indicates to perturb up to linear terms.*

Now apply the second deflation theorem 3.14 to get the 6×6 *system* $\boldsymbol{g} = (1.018\,x_1^2 + 1.071\,x_1 + (\varepsilon_{12} - 1.069)\,x_2,\ \varepsilon_{12} - .02023,\ .01723 + 2.036\,x_1,\ 1.058\,x_2^2 + (1.024 + \varepsilon_{23})x_1 + (\varepsilon_{22} - 1.016)x_2 + \varepsilon_{21},\ .04217 + 2.116\,x_2 + \varepsilon_{22},\ \varepsilon_{23} - .03921)$*, which has a regular root for* $\boldsymbol{\zeta} \in Z$ *and parameters* $(\varepsilon_{12}, \varepsilon_{21}, \varepsilon_{22}, \varepsilon_{23})$*. Indeed, applying Theorem 3.15 with* $Z' = Z \times [-.04, .04]^4$ *and* $(\boldsymbol{\zeta}^*, \mathbf{0})$ *we get a verified inclusion* $V_{\boldsymbol{g}}(Z', (\boldsymbol{\zeta}^*, \mathbf{0})) \subseteq \mathit{interior}(Z')$*.*

Example 3.21. *Consider the system [66] of* 3 *equations in* 2 *variables* $f_1 = x_1^3 + x_1x_2^2,\ f_2 = x_1x_2^2 + x_2^3,\ f_3 = x_1^2x_2 + x_1x_2^2$*, and the singular point* $(0, 0)$*.*

Suppose that the point is given. Using 3.6 and 3.8 we derive the primal-dual pair $\mathcal{D} = (1, d_1, d_2, d_1^2, d_1d_2, d_2^2, \underline{d_2^3} + d_1^3 + d_1^2d_2 - d_1d_2^2)$*, where* d_2^3 *is underlined to show that it corresponds to* x_2^3 *in the primal monomial basis* $\mathcal{B}$*. The biggest matrix used, in depth* 4*, was of size* 9×8*, while Macaulay's method terminates with a matrix of size* 30×15*.*

To deflate the root, we construct the augmented system $\mathcal{D}\boldsymbol{f}$ *of* 21 *equations. The* 21×2 *Jacobian matrix* $J_{\mathcal{D}\boldsymbol{f}}(\boldsymbol{x})$ *is of rank* 2 *and a full-rank minor consists of the rows* 4 *and* 5*. Therefore, we find the system* $(d_1^2[f_1], d_1d_2[f_1]) = (3x_1, 2x_2)$ *which deflates* $(0, 0)$*. Note that even though both equations of the deflated system derive from* f_1*, the functionals used on* f_1 *are computed using all initial equations.*

CHAPTER 4

Computing planar semi-algebraic sets

Contents

In the present chapter we present a new technique to handle semi-algebraic sets in the plane. The defining equations of the set are transformed to tensor-Bernstein form. This gives a numerically stable way to subdivide this representation into sub-domains, until certain regularity conditions are fulfilled. During the subdivision process the cells that touch the boundary of the semi-algebraic set are identified and their adjacency structure is represented as a graph. When this process terminates, we follow this graph to recover contours that define the geometry of the set. A tolerance $\varepsilon > 0$, given at the input, controls the precision of the computed approximation. Nevertheless, the regularity conditions (e.g. monotonicity of the function) imply a topologically correct result. In this sense, our algorithm, also published in [70], extends the approach in [2] on the arrangements of algebraic curves, by providing an efficient way to deal with semi-algebraic regions and to perform boolean operations on these regions.

In Section 4.2 we provide details on the representation of the main objects in memory. Then in Section 4.3 we describe a subdivision process that computes a collection of cells covering the semi-algebraic set. This representation is used to compute its connected regions, in Section 4.4. We specialize the main functions that appear in the algorithm first for the case of so called basic sets, in Section 4.5

and then discuss the general case in Section 4.6. We conclude with examples and an overview of our implementation in Section 4.7.

4.1 Planar semi-algebraic sets

Planar semi-algebraic sets are unions of subsets $\mathcal{S}$ of $\mathbb{R}^2$ that satisfy a set of bi-variate polynomial equalities and inequalities. These sets appear naturally when polynomial constraints are used for instance to describe regions of validity for a physical problem. Piece-wise algebraic representation of shapes is commonly used in CAGD, for instance in B-spline parametric representation of curves, or even surfaces of volumes, that also belong to the class of real semi-algebraic sets. Constructive Solid Geometry models also used in CAGD are semi-algebraic sets if the involved solid primitives are algebraic. In domains such as optimization, an important problem is the computation of global optimum of (polynomial) functions under (polynomial) constraints. These constraints define a semi-algebraic set as the solution space, in which the optimal points will be searched [62], [51]. In other words, semi-algebraic sets provide a general framework to handle many shape representations that are commonly used in shape modeling. For instance, computing the topology of an algebraic curve, or even the arrangement of several overlayed algebraic curves is a task that appears often in CAGD, and several effective methods have been proposed for this task [28].

The study of real semi-algebraic sets has a long historical background [101], with important theoretical contributions for instance on their triangulation [49], [52]. More algorithmic questions have also been tackled, essentially using the well-known Cylindrical Algebraic Decomposition [21]. This approach is based on performing successive projections of semi-algebraic sets onto sub-spaces of dimension one less and then lifting back to the projected set. It yields a decomposition of a semi-algebraic set S into (connected) components, defined by sign conditions deduced from some "sub-resultant" polynomial sequences [15], [23], [10], or from computing some generalized critical values of the equations [30, 92].

Matrix representations and operations on then, for basic families of such sets, eg. curves and surfaces, have been explored recently [68].

One of the bottlenecks for practical applications of C.A.D.-based approaches, even in small dimension, is its double exponential complexity behavior. This is due mainly to computations with algebraic numbers of possibly high degree. Other obstacles include the lack of extension to approximate computation, required by applications in CAGD and the problem of robust description of the components. Our approach refrains from costly algebraic manipulations, hence avoids the high complexity of exact computation. It is based on real root isolation, well suited for approximate yet certified computations. Moreover, it gives an answer to the

problem of representing the semi-algebraic set in a way that is both topologically correct and suitable for applications. This overcomes the inflexible description by sign conditions or other implicit descriptions, for instance the one in [9], where each connected component is described itself as a semi-algebraic set.

We note that our method can be extended to dimension three, without theoretical obstacles. Indeed, the implementation is done in a generic programming framework that allows extension to dimension three with relatively little additional effort, since abstract types and templated data structures are heavily used.

We start by defining the family of sets that we are interested in.

Definition 4.1. *The family* $\mathfrak{S} \subseteq 2^{\mathbb{R}^2}$ *of* semi-algebraic sets *is the closure under union and intersection of subsets of* $\mathbb{R}^2$ *of the form*

$$\left\{(x,y) \in \mathbb{R}^2 \,:\, f(x,y) = 0\right\} \quad \textit{and} \quad \left\{(x,y) \in \mathbb{R}^2 \,:\, g(x,y) > 0\right\}$$

where $f,\ g \in \mathbb{R}[x,y]$.

We call the above sets *basic semi-algebraic sets*. These definitions extend naturally to higher dimension.

If $\mathcal{S} \in \mathfrak{S}$, its complement $\mathcal{S}^c = \mathbb{R}^2 \setminus \mathcal{S}$ is easily seen to belong to $\mathfrak{S}$. The family $\mathfrak{S}$ is thus stable by intersection, union and complementary. Another important property of semi-algebraic sets is that the projection of a semi-algebraic set is a semi-algebraic set [10].

Our algorithm has as input an *initial frame* $\mathcal{D}_0 = [a,b] \times [c,d]$ and a semi-algebraic set $\mathcal{S}$, given in *disjunctive normal form*, that is, in the form $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$ where each $\mathcal{S}_i$ is an intersection of basic semi-algebraic sets, hence defined as a subset $\{(x,y) \in \mathbb{R}^2 \,:\, g_1 = 0, \ldots, g_m = 0,\ f_1 > 0, \ldots, f_n > 0\}$. It outputs a boundary effective representation of the connected components of this semi-algebraic set.

Given a precision $\varepsilon > 0$, it can also output a polygonal approximation of the set inside the domain $\mathcal{D}$, within the precision ε, which moreover is isotopic to S in the following sense:

Definition 4.2. *Two semi-algebraic sets* $\mathcal{S}_1, \mathcal{S}_2$ *of* $\mathbb{R}^2$ *are* isotopic *if there exists a continuous application* $F : \mathbb{R}^2 \times [0,1] \mapsto \mathbb{R}^2$ *such that* $F|_{t=0}$ *is the identity map,* $F(\mathcal{S}_1, 1) = \mathcal{S}_2$ *and for all* $t \in [0,1]$, $F|_t : \mathbb{R}^2 \mapsto \mathrm{Im}F|_t$ *is a homeomorphism.*

We introduce some notation. Throughout the text $\mathcal{S}$ will refer to an input semi-algebraic set. By a slight abuse of notation we might denote by $\mathcal{S}$ both the semi-algebraic set and the set of underlying defining polynomials. The meaning will be clear from the context. Let f be a polynomial of the input. We refer to parts of the real algebraic curve $f = 0$ that belong to $\partial\mathcal{S}$, the boundary of $\mathcal{S}$, as *boundary curves*. Points where boundary curves intersect (or a single boundary branch, part

of some $f = 0$ is self-intersecting), are called *crossing points*. Also, we will refer to a *branch* of a curve, defined by two endpoints $p,\ q$, as the part of the curve between these points, e.g. the image of a continuous parametrized curve $r : [0, 1] \to \mathbb{R}^2$ s.t. $r(0) = p$, $r(1) = q$ and $f \circ r = 0$.

4.2 Representation

We begin by describing the main objects in the algorithm, called hereafter cells, and how they are represented in memory.

A *cell* carries local information for $\mathcal{S}$ in a rectangular domain $\mathcal{D} = [a, b] \times [c, d]$. This information includes the Bernstein representation over $\mathcal{D}$ of the defining equations of $\mathcal{S}_i$, whenever $\mathcal{S}_i \cap \mathcal{C} \neq \varnothing$. It also carries the intersections of every branch of $\partial\mathcal{S}$ that crosses the cell with the cell frame $\partial\mathcal{C}$. The cells of interest are exactly the cells that contain branches of boundary curves, i.e. parts of $\partial\mathcal{S}$. These cells are identified during the subdivision process.

A local description of $\mathcal{S}$ in a cell is achieved using the tensor-Bernstein representation over $\mathcal{D}$ of every polynomial that defines $\mathcal{S}$. This representation is computed using DeCasteljau's algorithm. It yields, for a polynomial $f \in \mathbb{R}[x, y]$, an expansion

$$f(x, y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} \gamma_{i,j}\, B^i_{d_x}(x; a, b)\, B^j_{d_y}(y; c, d)\ , \tag{4.1}$$

where $d_x,\ d_y$ is the degree of f in $x,\ y$ resp., and $B^i_{d_x}(x; a, b)$ is the $i-$th Bernstein polynomial of degree d_x over the interval $[a, b]$, namely

$$B^i_{d_x}(x; a, b) = \binom{d_x}{i} (x - a)^i (b - x)^{d_x - i}\, (b - a)^{-d_x}\ , \tag{4.2}$$

$0 \leq i \leq d_x$, $a < b$. Consequently we store a $(d_x + 1) \times (d_y + 1)$ matrix in memory to represent f, i.e. a dense Bernstein representation. A number of properties of this basis, e.g. convexity, variation diminishing, positivity etc, make it suitable for stable approximate computations (see [44] for more information).

The first cell $\mathcal{C}$ that is computed as soon as the algorithm is launched is the one corresponding to the initial frame $\mathcal{D}_0$. This initial cell carries all the polynomials of the input. When a sub-cell is computed, if $\partial\mathcal{S}_i$ does not cross that cell, for some i, $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$, then the polynomials of $\mathcal{S}_i$ are not kept in the representation of it.

Another object is the *region*, which is a linear approximation of a 2-dimensional connected component of the semi-algebraic set. It is described as a collection of *contours*, that are closed loops properly oriented to delimit the region: The outer

contour, or *shell*, is oriented counter-clockwise (CCW for short) whereas any internal contours, or *holes* are clockwise (CW) oriented. See Fig. 4.4 for a region defined by three contours.

Every contour is essentially a simple polygon described as a list of vertices that lie on the boundary of the exact set.

We also employ graph structures to keep adjacency information between cells. These are internally saved in memory using adjacency-list representation [22].

More specifically, we compute an undirected graph $\mathcal{A}$, in which the points where $\partial\mathcal{S}$ intersects $\partial\mathcal{C}$ correspond to edges and subdivision cells $\mathcal{C}$ correspond to vertices. We shall compute the restriction of the semi-algebraic set in a given initial domain, thus the border of this domain is from a computational point of view a limit for the regions to compute. For this reason, we also keep a directed graph containing the cells where boundary curves touch the initial frame and the four corner cells of $\mathcal{D}_0$. This forms a CCW loop and is used to complete any open contours that touch the boundary.

The space subdivision is tracked using a $kd-$tree, rooted at $\mathcal{D}_0$. The leaves of this tree is a partition of $\mathcal{D}_0$ into cells. The inner nodes represent the sequence of subdivisions that took place.

Example. In Fig. 4.3(left), we have a partition of the domain into 8 regular cells. The semi-algebraic set is the grayed area, described by a single contour. Here the graph $\mathcal{A}$ is the closed path of cells 2,7,6,8,3,2. The border graph is the directed closed path 2,1,7,6,4,3,2.

4.3 Subdivision process

The subdivision of the initial domain into regular cells is a main operation of the algorithm. It consists in splitting the initial domain into smaller cells until certain local properties are satisfied. These properties will allow in a later step the construction of a topologically correct approximation of the (boundary of the) set in each cell.

During this process we construct a graph $\mathcal{A}$ whose vertices are the cells that span $\partial\mathcal{S}$. Alg. 4.1 presents the general process. Here a cell is regarded as an abstract object that supports the following operations:

• **Regularity test**(IsRegular). A cell is considered regular if the topology of $\mathcal{S}$ inside the cell is known, i.e. it can be deduced using only discrete data stored in the cell, namely the points in $\partial\mathcal{C} \cap \partial\mathcal{S}$, or even the sign of some derivatives on them. Hence interesting cases are the cells that contain branches of boundary curves. Some characteristic examples of this are presented in Fig. 4.1. If there is more than one crossing point in the cell, that is, branches that intersect each other, then there is ambiguity on how the region behaves in the cell. Thus the regularity

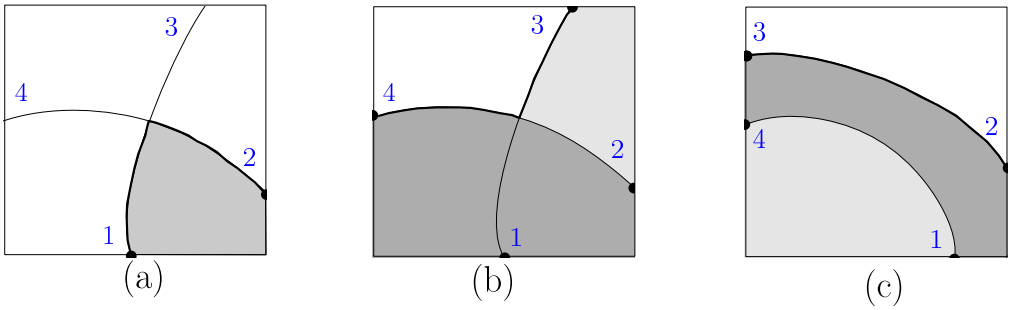

Figure 4.1: Examples of cells that are regular and intersect $\mathcal{S}$: (a) intersection of two basic sets, (b) union of two basic sets with a crossing, (c) union of two sets.

implies that we have at most one crossing point inside the cell and that the branches inside $\mathcal{C}$ have a monotone behavior. This behavior is connected to special points on the boundary curves, namely points with vertical or horizontal tangents.

• **Boundary curve intersection test**(ONBOUNDARY). It is used to identify if a cell is intersecting $\partial\mathcal{S}$, i.e. $\mathcal{C} \cap \partial\mathcal{S} \neq \varnothing$. This can be done by inspecting the sign variations of Bernstein coefficients of the polynomials that define $\mathcal{S}$. *Descartes' Rule of Signs* implies that if there is a branch of $\partial\mathcal{S}$ in $\mathcal{C}$, then there will be sign variations on the coefficients of some boundary equation. On the other hand, by the positivity property of the Bernstein basis, if the coefficients over a cell $\mathcal{C}$ of a curve $f = 0$ have no sign variations, then there cannot be a branch of this curve in $\mathcal{C}$.

Algorithm 4.1: Subdivision algorithm

Input: A cell $\mathcal{C}_0$ corresponding to the initial domain $\mathcal{D}_0$.
Output: A partition of $\mathcal{C}_0$ into regular cells and a cell graph $\mathcal{A}$.
Initiate a $kd-$tree $\mathcal{K}$ and set its root to $\mathcal{C}_0$;
Initiate a graph $\mathcal{A}$ with a vertex $\mathcal{C}_0$;
for *all unvisited leaves* $\mathcal{C}$ *in* $\mathcal{K}$ **do**
  if ONBOUNDARY$(\mathcal{C})$ *and not* ISREGULAR$(\mathcal{C})$ **then**
    subdivide $\mathcal{C}$ into two children $\mathcal{C}_L$ and $\mathcal{C}_R$;
    put an edge in $\mathcal{A}$ between $\mathcal{C}_L$ and $\mathcal{C}_R$;
    distribute the $\mathcal{A}-$neighbors of $\mathcal{C}$ to $\mathcal{C}_L, \mathcal{C}_R$;
    remove $\mathcal{C}$ from $\mathcal{A}$;
  else
    mark $\mathcal{C}$ as visited;
  end
  return $\mathcal{K}$, $\mathcal{A}$;
end

During the subdivision process the following information is computed:

- ◇ Space partition information in the kd-tree structure.
- ◇ Local information in the subdivided cells: the tensor-Bernstein representation over the cell, critical points contained in the cell, intersection points of $\partial\mathcal{S}$ with the cell frame.
- ◇ Adjacency information between the cells, in horizontal and vertical direction. The cells in which the boundary curves touch the border $\partial\mathcal{D}_0$ are also connected in a counter-clockwise loop, to serve the purpose of limiting the computation inside $\mathcal{D}_0$.

• **Space Partition.** The cells that derive from successive subdivisions are organized in a kd-tree structure [12], rooted at the initial domain $\mathcal{D}_0$. The nodes in this tree have pointers to their left and right children, as well as to the parent node. The coordinate in which the subdivision takes place at every level of the tree is not fixed; it is implied every time by the dimensions of the current cell, thus at the same level of the tree we may have cell subdivisions either in x or y coordinate.

This structured partition allows to perform fast point location queries. The reason we have chosen a kd-tree rather than a quad-tree is economy wrt the overall number of cell subdivisions as well as the modularity that it offers, for instance it's direct adaptation to three or more dimensions.

There are two basic tests to be defined, to guide the subdivision process. The first identifies that a cell is *regular*, i.e. the topology of the semi-algebraic set in the cell is known. In this case the subdivision stops at this branch of the kd-tree. The second test identifies if $\partial\mathcal{S}$ intersects the current cell. If not, then either $\mathcal{C} \subseteq \mathcal{S}$ or $\mathcal{C} \cap \mathcal{S} = \varnothing$, thus there is no need to subdivide it any further.

• **Cell subdivision.** Subdividing a cell $\mathcal{C}$ along some coordinate is essentially to compute, starting from the Bernstein representation over $\mathcal{C}$, representations over some sub-domains of $\mathcal{C}$. This operation is carried out by one call of DeCasteljau's algorithm [44].

Moreover, along the line where the splitting takes place, we solve a univariate Bernstein polynomial for every boundary curve that intersects the cell, in order to compute intersection with the new frame sides. The existing crossing points and frame intersection points are distributed to the resulting sub-cells, Fig. 4.2.

• **Adjacency graph update.** At each subdivision step, a former leaf of the kd-tree obtains two children. To update the cell graph, we disconnect this node and distribute it's neighbors to the new children, according to the direction of splitting. Finally, we introduce a new edge that joins the two children along the corresponding direction. These steps, demonstrated in Fig. 4.2, ensure that at any point of the subdivision, the leaves of the kd-tree, which form a partition of $\mathcal{D}_0$, are connected to the neighboring cells in all four sides.

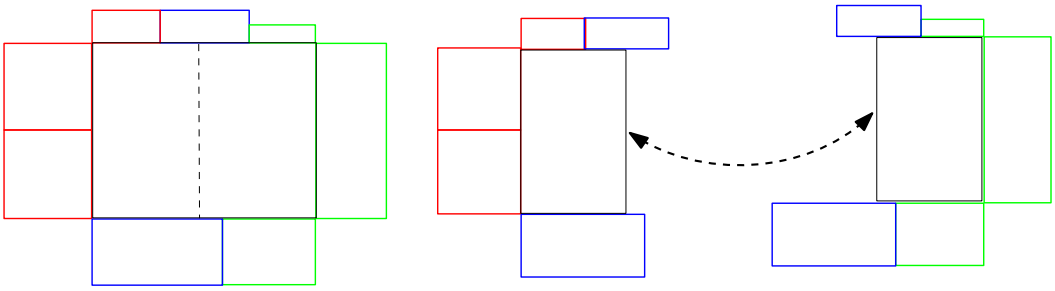

Figure 4.2: Cell subdivision along $x-$direction. Neighbors of the parent (left) are distributed to the children(right). An edge is added between the latter.

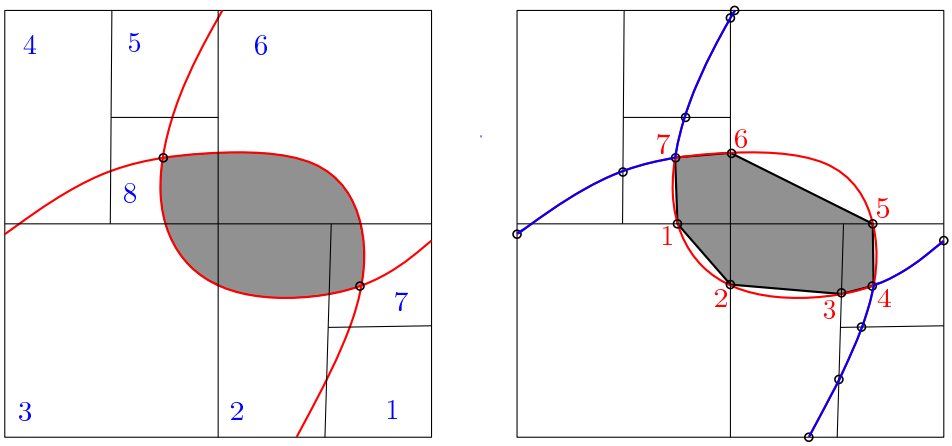

Figure 4.3: Left: Subdivision process, with marked subdivided cells and intersections. Right: Computed polygonal region, marked with the oriented list of contour points.

4.4 Region recovery

In this section we explain how we pass from the cell description to a polygonal approximation of the (connected components of the) semi-algebraic set. We will demonstrate that as soon as the subdivision Alg. 4.1 terminates, we are able to recover the shape of the semi-algebraic set, and guarantee the correctness of the construction.

The output is a list of regions that correspond to connected components of the semi-algebraic set. The set of cells that intersect a region can readily provide a triangulation of the region, which can be outputted for use in rendering. Each region is represented as a set of closed oriented contours. The orientation of every contour reveals whether it is the exterior boundary, or *shell* of the region, or an internal gap, or a *hole*. There is a unique shell for every region of $\mathcal{S}$.

To compute the regions, it suffices to traverse the cell graph $\mathcal{A}$ in a suitable way and recover the shell and holes of every region in the set. The algorithm for region computation is summarized in Alg. 4.2.

Algorithm 4.2: Region computation

Input: A cell graph $\mathcal{A}$ covering the semi-algebraic set $\mathcal{S}$.
Output: A list L of polygonal regions, one for every connected component of $\mathcal{S}$.

```
L ← ∅;
for all boundary cells C in A do
    if C is not visited then
        F ← DiscoverContour(C);
        if F IsCCW then
            Initialize region R with F;
            push R to L;
        else
            attach hole F to it's containing shell
        end
    end
end
return L;
```

The orientation check IsCCW depends only on the contour F. Every closed contour can be assigned an orientation; if one walks around the curve in such a way as to keep the bounded region on one's left at all times, the contour is said to be positively oriented. If the contour is traversed in the opposite direction, then it is said to be negatively oriented. Let $c = (p_1, p_2, \ldots, p_n)$ with $p_i = (x_i, y_i)$, $p_{n+1} = p_1$ be a list of points defining a closed polygonal contour. The sign of

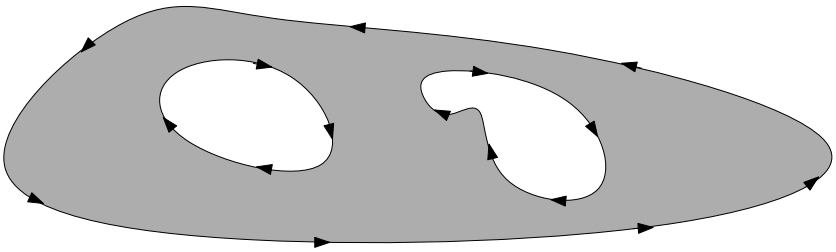

Figure 4.4: A region defined by it's oriented border. All the contours are CCW-oriented wrt the grayed region. This leaves the holes CW oriented with respect to the bounded domain they define.

the quantity $\sum_{i=1}^{s}(x_i y_{i+1} - x_{i+1} y_i)$ determines whether c is positively or negatively oriented. This sum is twice the (signed) area of the contour.

The function DISCOVERCONTOUR, presented in Alg. 4.3, returns a contour that crosses the cell $\mathcal{C}$ and is oriented CCW wrt the region it delimits. For instance, both the holes and the shell of the region in Fig. 4.4 are CCW oriented wrt the grayed region. It is required that the cell argument is *regular*, so that the global shape of the contour can be determined by following the known local topology in the cell. This is ensured by the subdivision process of Section 4.3.

Apart from the cells containing branches of the boundary contours, there are special cells that are needed in order to constrain the computation in the initial frame $\mathcal{D}$. These are the boundary cells that touch the frame as well as the four corners of $\mathcal{D}$. They are connected in a CCW loop during the subdivision process that is used to complete the contours that escape $\mathcal{D}$ and would not be closed otherwise.

4.4.1 Following the boundary curves around a region

The main function in Alg. 4.2 is DISCOVERCONTOUR, which is in turn based on two routines, PAIR and STARTINGPOINT.

• **Pair.** If a cell intersects both a region and the region's boundary, then for every intersection point p there is a unique point q that is connected to p via a segment of $\partial\mathcal{S}$ that lies inside the cell. If the cell in question is also regular, q can be computed using sign conditions along $\partial\mathcal{C}$. We define this point q to be the result of PAIR$(\mathcal{C}, p)$. If this point q is different from p, then evidently it is connected to p via a branch of some boundary contour of the region. Alg. 4.4 presents a general strategy to compute q.

Example. In Fig. 4.1 the result of PAIR is: (a)$1 \to 2$, (b)$4 \to 3$, (c)$2 \to 3$. Note that in case (c), the branch $1 \to 4$ will not occur in the computation, since it does not belong to $\partial\mathcal{S}$.

• **Starting Point.** A contour has to be traversed with the correct orientation, otherwise we would not be able to distinguish between shells and holes of a region. For this, it suffices to provide the first two points in the point list of the contour with the correct orientation. This is the task of the STARTINGPOINT($\mathcal{C}$) routine. It returns a point p on $\partial\mathcal{C}$ s.t. the oriented branch with endpoints p, PAIR(p) has on its left side the region to be computed. This is a special case of the PAIR computation described in the next paragraph.

Example. For Fig. 4.1 the result of STARTINGPOINT is: (a)2, (b)3, (c)2. Indeed, the respective branches (a)$2 \to 1$, (b)$3 \to 4$ and (c)$2 \to 3$, are CCW-oriented wrt $\mathcal{S}$.

Looking at the graph $\mathcal{A}$ induced by Alg. 4.1, we distinguish two kinds of regular cells:

◇ Cells that contain non-crossing branches of $\partial\mathcal{S}$.

◇ Cells that contain branches that intersect at one crossing point.

Recall that the outcome of PAIR($\mathcal{C}$) is the point connected to p via a branch which lies inside $\mathcal{C}$. The general algorithm is presented in Alg. 4.4. The essential tool for this computation is an efficient way to check if a given point on $\partial\mathcal{C}$ is contained in $\mathcal{S}$. This is done using the sign of the Bernstein coefficients. For every polynomial f of $\mathcal{C}$, there are four *extreme* coefficients that are equal to it's value on the four corners of $\partial\mathcal{C}$. Now taking into account that the sign of f along $\partial\mathcal{C}$ alternates every time we pass a boundary intersection point, we can determine the sign on any point of $\partial\mathcal{C}$ by starting from an extreme coefficient and counting points along $\partial\mathcal{C}$, up to the desired point.

If there is one crossing point in $\mathcal{C}$ the topology of $\partial\mathcal{S} \cap \mathcal{C}$ is conic (Fig. 4.1(a,b)). To choose the correct pair of a given point on $\partial\mathcal{C} \cap \partial\mathcal{S}$, we check whether a $\partial\mathcal{C}-$neighborhood on the left of p belongs to $\mathcal{S}$ or on the right of p. We output accordingly the point on the side where the test was positive.

If there is no crossing point, (Fig. 4.1(c)) it suffices to return the other end of the branch that starts from p. We shall see in the sequel how this information is recovered on regular cells.

Example. In the case of Fig. 4.3(left) we execute Alg. 4.3. Starting from cell 2, we obtain a first point of the contour, using STARTINGPOINT routine. Succesive calls of the pair function give the sequence of points shown in Fig. 4.3(right). The process stops when we reach the cell 2 again, thus completing the contour.

It remains to specialize these functions. We continue by doing so, first in the case of basic algebraic sets and then in the case of intersection and union.

Algorithm 4.3: DISCOVERCONTOUR($\mathcal{C}$)

Input: A *regular* cell $\mathcal{C}$ of $\mathcal{A}$.
Output: A list F of points in the plane that define a closed contour.
$p \leftarrow$ STARTINGPOINT($\mathcal{C}$);
Initialize a contour F and push p to it;
$\mathcal{C}_0 \leftarrow \mathcal{C}$;
repeat
  mark $\mathcal{C}$ as visited;
  $p \leftarrow$ PAIR($\mathcal{C}, p$);
  push p to contour F;
  $\mathcal{C} \leftarrow$ the $\mathcal{A}-$neighbor of $\mathcal{C}$ that contains p;
until $\mathcal{C} = \mathcal{C}_0$;
return F;

Algorithm 4.4: PAIR

Input: A regular cell $\mathcal{C}$ and an intersection point p on $\partial\mathcal{C}$.
Output: The intersection point q such that $\{p,\ q\}$ lie on a branch of $\partial\mathcal{S}$.
if *there is a crossing in* $\mathcal{C}$ **then**
  Let l, r be the CCW previous and next point, resp., of p, in $\partial\mathcal{C} \cap \{f = 0\}$;
  Based on which of the segments $\overline{lp}$ or $\overline{pr}$ lies in $\mathcal{S}$, return either l or r ;
else
  return the other end of the $\mathcal{C}-$branch starting from p;
end

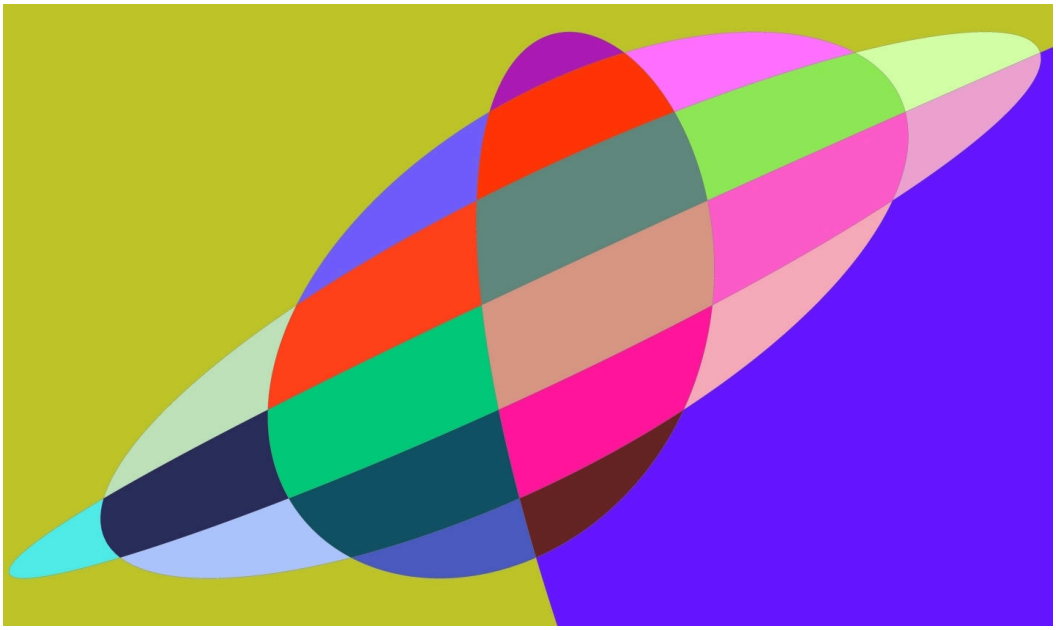

Figure 4.5: The 23 faces in the topology of the degree 8 curve $f = 2 + 7x - 7y - 14x^3 + 7x^5 - x^7 - 16y^2 + 14y^3 + 20y^4 - 7y^5 - 8y^6 + y^7 + y^8 - 42y^2x - 70y^3x^2 + 35xy^4 + 70y^2x^3 + 42yx^2 - 35x^3y^4 + 7x^6y - 21x^5y^2 - 35x^4y + 21x^2y^5 + 35y^3x^4 - 7xy^6$ computed by running our algorithm on $\mathcal{S} = \{f > 0\}$, $\mathcal{S} = \{f < 0\}$ and $\mathcal{D} = [-4, 4] \times [-3, 3]$.

4.5 The case of basic semi-algebraic sets

A basic semi-algebraic set is defined by one polynomial, $\mathcal{S} = \{(x, y) : f > 0\}$, or $\mathcal{S} = \{(x, y) : f = 0\}$. In both cases the treatment is quite the same, and depends on the boundary curve $f = 0$, hence we shall suppose $\mathcal{S} = \{(x, y) : f > 0\}$. In the case of equality it is only the contour lines that will be outputted rather than two-dimensional regions. After fully treating this case, we shall generalize by extending the operations to the cases of intersection and union.

This case is closely related to the topology computation of an implicit real algebraic curve. The latter is the partition of space into points, edges and faces defined by the curve $f = 0$. See Figure 4.5 for an example. Note that recovering the topology of the real algebraic curve $f = 0$ is a special case of our algorithm. Indeed, it suffices to execute the subdivision algorithm on $\mathcal{S} = \{f = 0\}$ and then run the region recovery twice, once with $\mathcal{S} = \{f > 0\}$ and once with $\mathcal{S} = \{-f > 0\}$. The union of these two outputs is exactly the set of faces defined by the curve $f = 0$.

4.5.1 Regularity test

We describe the regularity criteria that are used for the boundary curve of the set. We shall provide a brief overview of known techniques to arrive to regularity conditions that assure verified determination of the topology of boundary curves. Regular algebraic segments have been extensively studied, we refer to [2, 7, 111, 112] for details on their approximation and use in modeling.

The regularity depends on special points on the curve, that reveal the local shape of the curve in a neighborhood around them. These are:

Definition 4.3. *The set of* extremal points *of* $f \in \mathbb{R}[x, y]$ *is the solutions of the system* $\partial_x f(x, y) = \partial_y f(x, y) = 0$.
The set of singular points *of* f *is the subset of extremal points that also satisfy the equation* $f(x, y) = 0$.
The set of x-critical *(*y-critical*) points of* f *is the solution set of* $\partial_x f(x, y) = f(x, y) = 0\}$ *(*$\partial_y f(x, y) = f(x, y) = 0\}$*).*

Computing these points, approximately but also efficiently, is a vital ingredient of the algorithm. For this task, we rely on subdivision techniques (Chapter 2), eg. [80], which is the variant that works on Bernstein polynomials. This provides good approximations of the points in Def. 4.3. These points are precomputed and during the subdivision process they are isolated between the cells, i.e. we do not allow more than one of them in a single cell. As a result, after the subdivision process terminates, we obtain a partition of $\mathcal{D}_0$ into regular cells of the following type:

- ◇ *x-regular* cells, those that contain no x-critical points (similarly for $y-$regular).
- ◇ *simply singular* cells, that contain a single singular point and all branches of $\partial\mathcal{S} \cap \mathcal{C}$ intersect it.

• **Regular cells.** If a cell is $x-$regular, it contains a number of $x-$monotone branches. In short, the direction of the tangential gradient vector $(\partial_y f, -\partial_x f)$ evaluated at the points in $\partial\mathcal{C} \cap \partial\mathcal{S}$ yields the connection of the branches inside $\mathcal{C}$. The Bernstein representation of the derivatives themselves are easily computed, since they are given as differences of Bernstein coefficients of f. A sufficient condition for f to be x-regular is that the Bernstein coefficients of $\partial_x f$ maintains a constant sign. By Descartes' law, this statement implies that the sign variations in $x-$direction should be at most one.

Note that in special cases where the critical point is on $\partial\mathcal{C}$ two branches may share a starting or ending point.

• **Simply singular cells.** If there is a single singular point in a cell $\mathcal{C}$, and no additional extremal points, one must test whether all the branches inside $\mathcal{C}$ cross

this point. This would imply that the topology inside $\mathcal{C}$ is a cone starting from the singular point. The test is based on computing the topological degree, or Gauss map [98] of the vector field $\nabla f = (\partial_y f, \partial_x f)$ around the closed curve $\partial\mathcal{C}$. This breaks down to isolating the real roots of $\partial_x f$ and $\partial_y f$ along $\partial\mathcal{C}$. Khimshiashvili's theorem [57] relates the number of branches that reach the singular point to the topological degree $\deg(\nabla f, \mathcal{C})$; it states that the number of branches is exactly $2\,(1 - \deg(\nabla f, \mathcal{C}))$. If this number coincides with the cardinality of $\partial\mathcal{C} \cap \partial\mathcal{S}$ then we can treat this cell, otherwise there are additional branches in $\mathcal{C}$ and the subdivision will continue until they are isolated from the singular point.

4.6 The general case

To treat semi-algebraic sets with more than one defining equation/inequality, it suffices to extend the main operations in this case. Our aim is to have a covering of the boundary curves of $\partial\mathcal{S}$ by regular cells. The main difference is that crossing branches in a cell can correspond to two basic sets in a union, or two basic sets in an intersection. Treating correctly these cases will extend our algorithm to the whole family of semi-algebraic sets. Again, we assume that the basic sets are defined by inequalities, since restricting to (in the case of intersection) or attaching (in the case of union) a curve segment to the output is not essentially different from treating boundary curves of two dimensional components. In particular, the cell graph $\mathcal{A}$ that we obtain from the subdivision Alg. 4.1 will span any components of lower dimensions.

Let $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$. Recall that a cell $\mathcal{C}$ carries the polynomials of $\mathcal{S}_i$ if $\partial\mathcal{S}_i \cap \partial\mathcal{C} \neq \varnothing$. For all the other parts $\mathcal{S}_j$, it is either $\mathcal{S}_j \cap \mathcal{C} = \varnothing$ or $\mathcal{C} \subseteq \mathcal{S}_j$, hence $\mathcal{C}$ does not interfere with the boundary curves of these components.

We define a regular cell to be a cell in which every attached polynomial is regular (in the sense of Section 4.5.1) and conforms to any of the following properties:

1. There is only one set $\mathcal{S}_i$ in $\mathcal{C}$ and at most one (self-)intersection.

2. There are two sets $\mathcal{S}_i$ and $\mathcal{S}_j$ and one intersection between a branch of $f \in \mathcal{S}_i$ and $g \in \mathcal{S}_j$.

These intersection points are also computed by subdivision solving and are isolated among the cells during the subdivision process.

Deciding if a region spans $\partial\mathcal{S}$ is done by checking whether it belongs to the boundary of every $\mathcal{S}_i$ that is carried by $\mathcal{C}$, and consists again in checking signs on the boundary.

To simplify the process, we rely on basic cells (cells that have branches of a single basic set contributing to $\mathcal{S}$) for determining the orientation of regions, i.e.

applying STARTINGPOINT. This is a mild assumption, since in any case, boundary curves away from crossings define basic semi-algebraic sets. This assumption also simplifies the way we deal with cells like Fig. 4.1(c), since we only need to know the connection inside the cell in order to traverse them and choose the correct branch (for instance, in Fig. 4.1(c), discard the locally redundant curve).

We describe how we compute PAIR in the above two cases:

• **Case 1.** There is a set of branches in the cell that intersect in one point only, similar to 4.1(b). Since the corresponding basic sets are combined by intersection we search around $\partial\mathcal{C}$ for a part that attains positive sign on all involved polynomials, to decide the PAIR routine.

• **Case 2.** Two branches intersect, corresponding to basic sets combined by union, for instance 4.1(c). We propagate the search to points around parts of $\partial\mathcal{C}$ that satisfy any of the sign conditions implied by $\mathcal{S}_i$ or $\mathcal{S}_j$. When we reach a part that is outside $\mathcal{S}$, we return the last point found.

4.7 Implementation and demonstration

Our implementation is generic, working on abstract classes of cells, that define internally a small number of predicates. We chose to use the open-source project MATHEMAGIX [106], for the fast data structures it provides for polynomials and its support to certified arithmetic primitives. Our code is written in the frame of the `shape` package, which is the part of MATHEMAGIX providing a variety of geometric operations in two or three dimensions.

Solution of univariate and bi-variate systems of polynomial is performed using subdivision on the Bernstein representation, which is present in the package `realroot`. This library also provides algebraic operations, Bernstein dense representation and a variety of zero-dimensional system solvers. Hardware accelerated rendering of output has been made possible using AXEL* platform.

A first example is given in Fig. 4.6, where we can see the cells deduced by the subdivision process together with the defining curves (left), and the computed regions (right) based on this cell graph. The boxes span only the actual boundary curves of $\partial\mathcal{S}$, but we also draw the full defining curves to give an idea of the situation.

A precision of $\varepsilon = 0.05$ is used, that is, the cells are subdivided down to this size, to obtain a smooth visual result. Note the two branches that are almost tangent near the bottom left corner. They cause the subdivision to continue further around this area until the branches are properly separated.

In Fig. 4.7 we compute a set $\mathcal{S} = \{(x, y) : f_1 > 0, f_2 > 0\}$ defined by a degree 6 and a degree 32 polynomial. The domain of computation is $[-1.5, 1.5]^2$

*http://axel.inria.fr

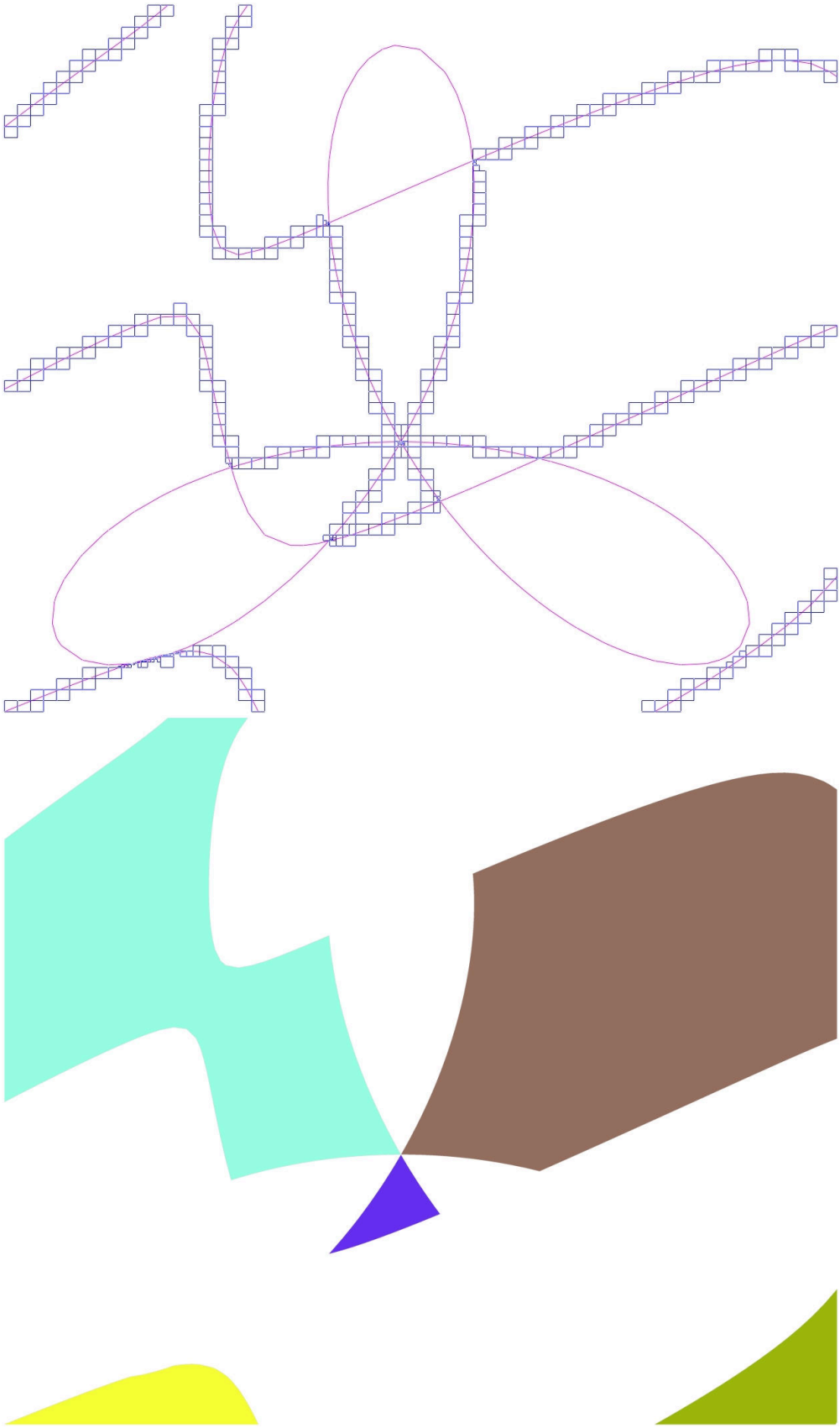

Figure 4.6: $\mathcal{S} = \{(x, y) : f_1 > 0, f_2 > 0\}$ with $f_1 = x^4 + 2x^2y^2 + y^4 + 3x^2y - y^3$, $f_2 = -105y^2x^4 - 80y^3 + 140x^3y^3 - 140y^3x + 35y^4 - 105y^4x^2 + 48y^5 + 42xy^5 - 42x^2 + 35x^4 - 7x^6 + 32y + 84xy - 140x^3y + 42x^5y + 210x^2y^2 - 42y^2 - 7y^6 - 8y^7 + 7$ over the box $[-1, 1]^2$.

Figure 4.7: Left: defining curves and cell graph. Right: boundary contours of the underlying set.

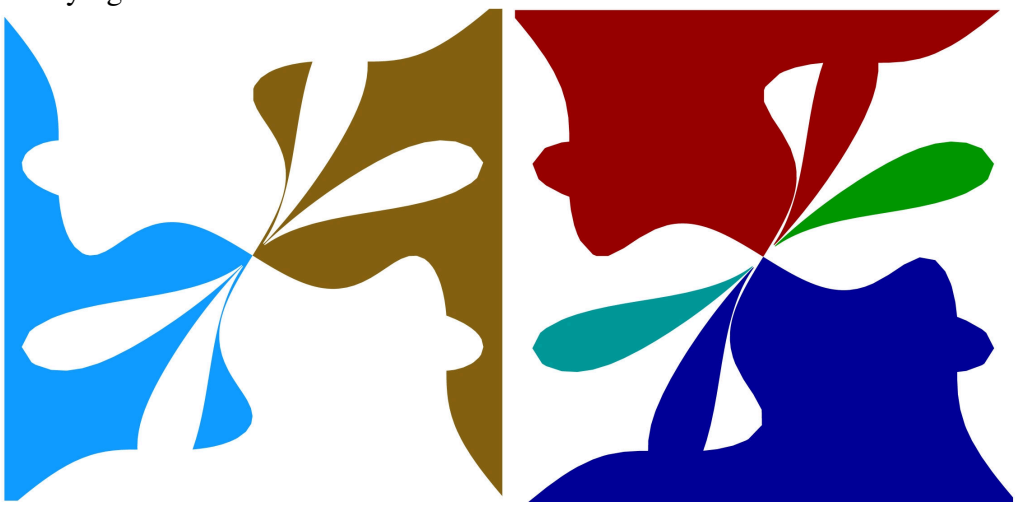

Figure 4.8: Left: Two connected components of a semi-algebraic set, each containing a hole. Right: regions of the complementary set.

and precision set as before, $\varepsilon = 0.05$. The running time for this example is less than one second. Our implementation is able to handle polynomials of quite higher degree, up to 100 or more. Here the resulting regions contain holes, which are correctly recognized. Finally, Fig. 4.7 presents the complementary set, $\mathcal{S}^c = \{(x, y) : -f_1 > 0\} \cup \{(x, y) : -f_2 > 0\}$ given by 4 connected components.

The purpose of the third example is to demonstrate how our implementation can handle degenerate cases, namely cusps. We treat a single curve of degree 28, having several cusps. This curve is taken from a real application in non-linear computational geometry, namely the computation of the Voronoi diagram of ellipses, see recent paper [43]. We compute all regions defined by the curve, in the domain $[-7, 3]^2$ and set precision to $\varepsilon = 0.5$. Detailed output is shown in Fig. 4.9. This computation was done in under 3 seconds.

More examples follow in Figures 4.10, 4.11, 4.12 and 4.13. The degree goes up to 76 for Figure 4.11, and the computation took less than five seconds.

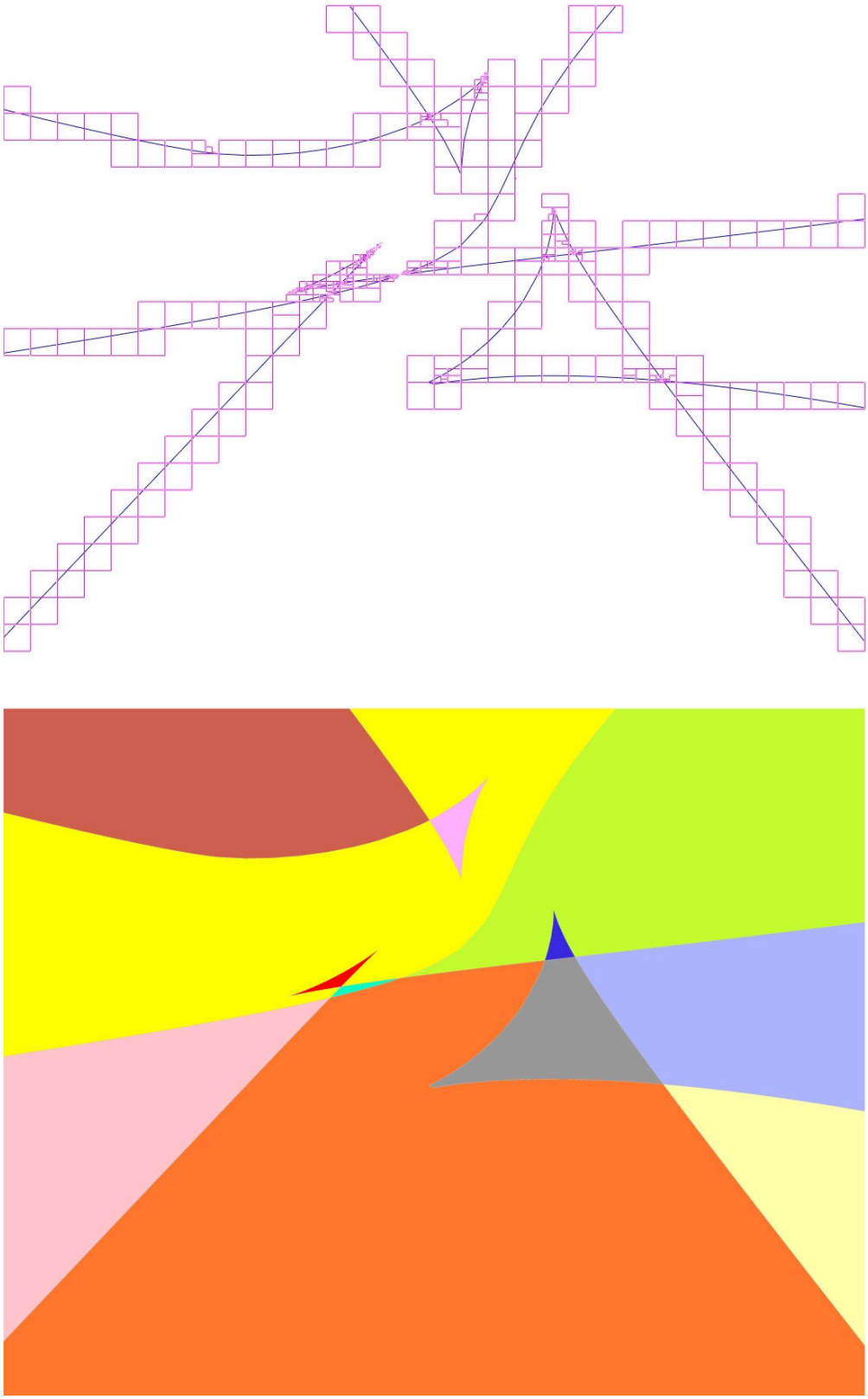

Figure 4.9: Computing the topology of a degree 28 algebraic curve with cusps.

Figure 4.10: Semi-algebraic set defined by: $f_1 = -105y^2x^4 - 80y^3 + 140x^3y^3 - 140y^3x + 35y^4 - 105y^4x^2 + 48y^5 + 42xy^5 - 42x^2 + 35x^4 - 7x^6 + 32y + 84xy - 140x^3y + 42x^5y + 210x^2y^2 - 42y^2 - 7y^6 - 8y^7 + 7$, $f_2 = x^2 + 3y^2 - 1$, $f_3 = x^6 + y^2x^4 - y^4x^2 - 2x^4 - y^6 + 2y^4 + x^2 - y^2 + xy$ in domain $[-3, 3]^2$.

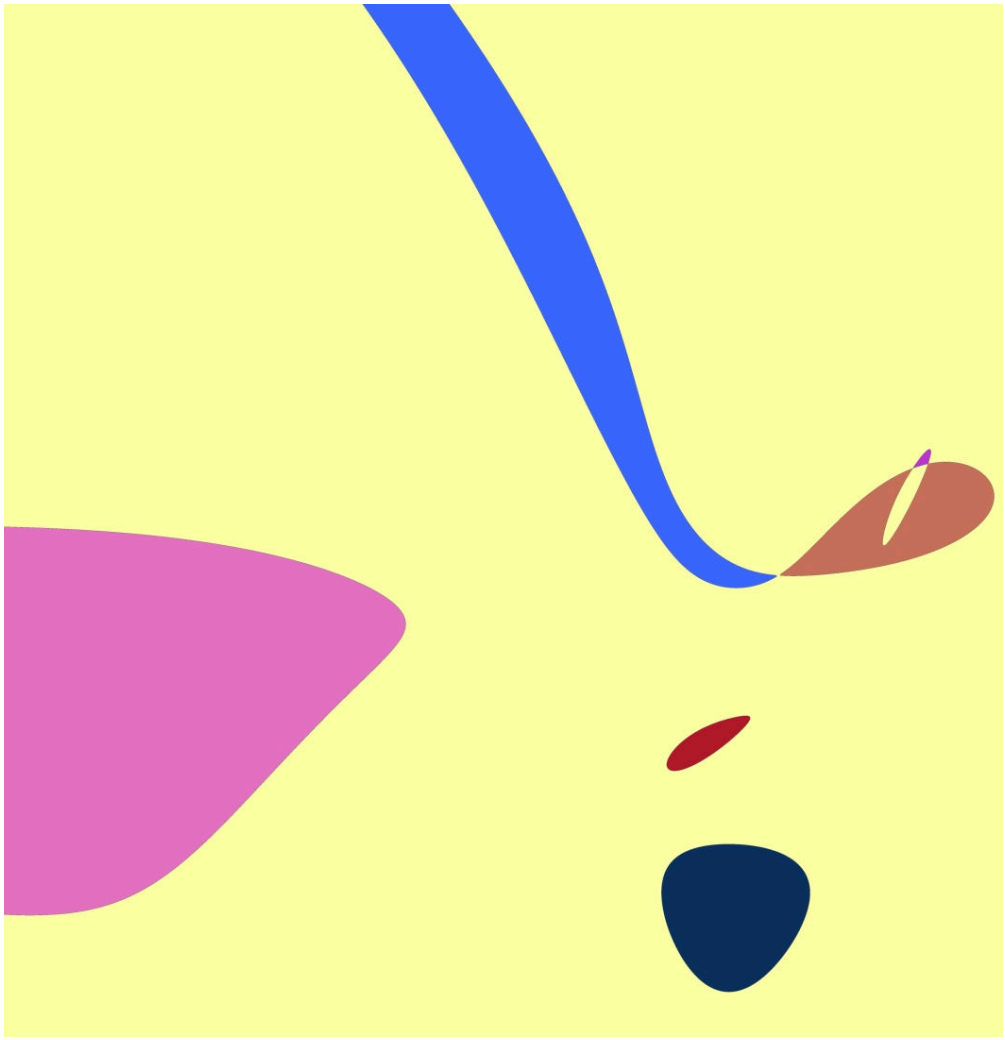

Figure 4.11: Topology of a degree 76 curve coming from the self-intersection locus of a 3D surface.

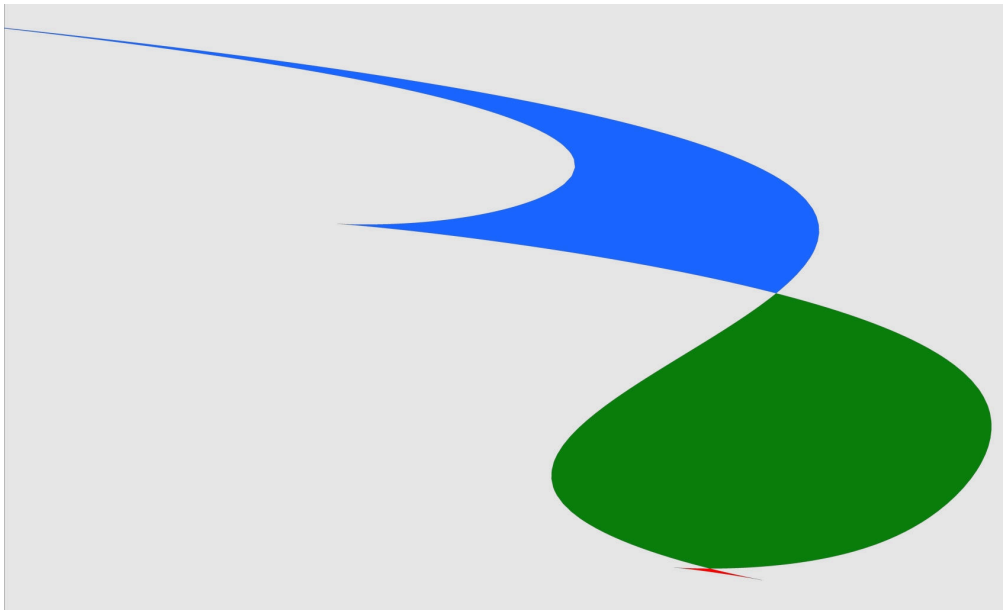

Figure 4.12: A (degree 12) apparent contour of 3D surface with cusps.

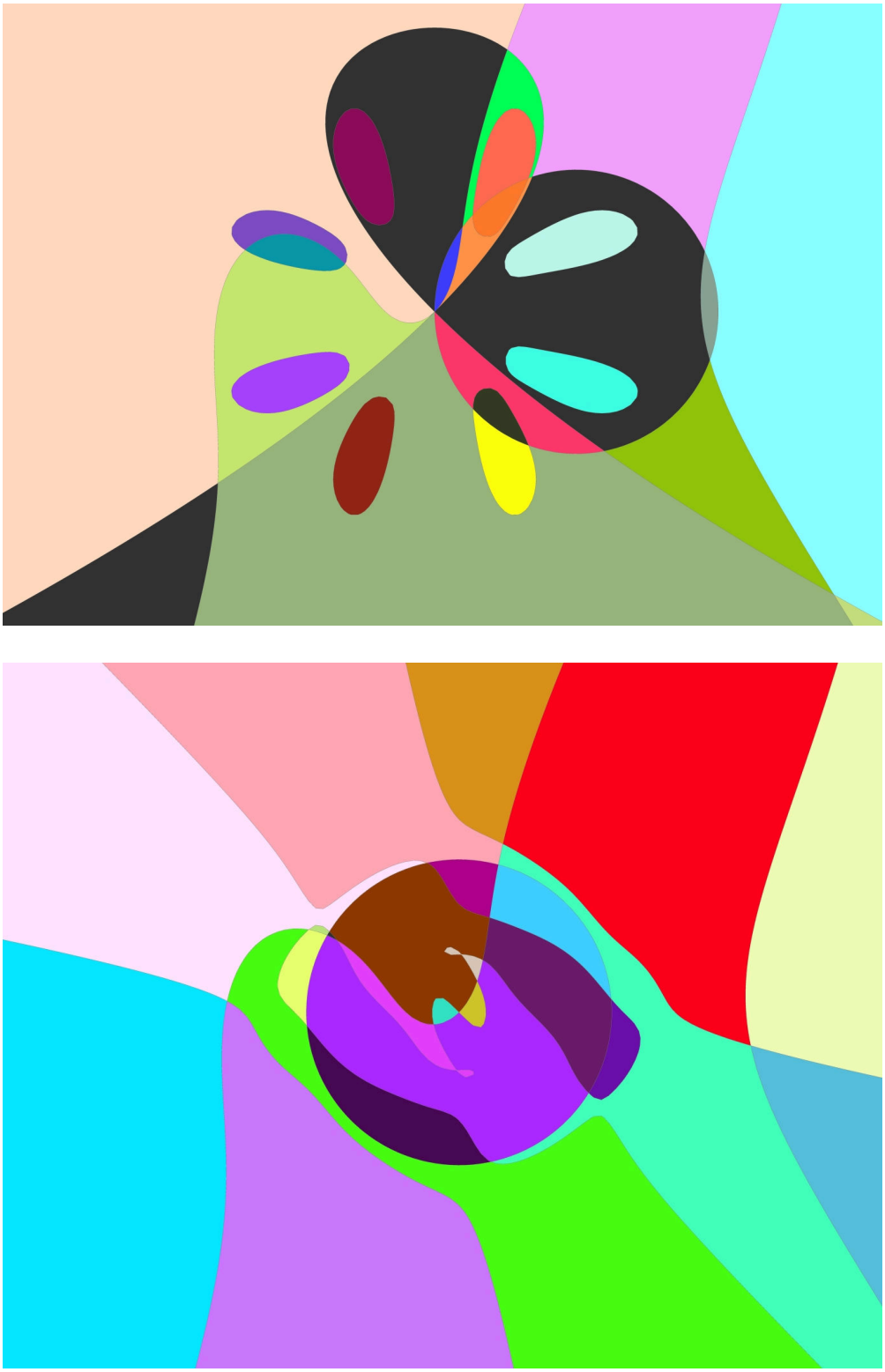

Figure 4.13: Regions in the arrangement of three curves, of resp. degrees 32,4,4(top), 32,4,13 (bottom) computed using our algorithm on the underlying semi-algebraic domains.

CHAPTER 5

Algebraic framework for generalized Voronoï diagrams

Contents

We design and implement a new algorithm for the computation of general Voronoi Diagrams (VD's) constrained to a given domain. Our method is applicable to any VD type in which the distance from a site can be expressed by a bi-variate polynomial function, notably the anisotropic VD or even VD's of complex sites. We use the Bernstein form of polynomials and DeCasteljau's algorithm to subdivide the initial domain and isolate bisector domains or domains that may contain a Voronoi vertex. The efficiency of our algorithm is due to a filtering process based on bounding the distance functions over the subdivided domains. This allows to exclude functions (thus sites) that do not contribute locally to the lower envelope of the lifted diagram. After the filtering process the bisector curves are approximated by line segments, and vertices are computed using a Bernstein solver, giving overall certified polygonal description of each Voronoi cell. Results presented in what follows have lead to an article, which is at the time of writing submitted (cf. [36]) for publication.

We discuss some background and previous work, and then we give a list of VD's, as well as some details on curve representations in Section 5.1.2. We present the core of our algorithm in Section 5.2, with details on the filtering, subdivision and the recovery phase. Then we extend the method to implicitly given distance fields in Section 5.3.

5.1 Introduction

Voronoi Diagrams (VD's) have a surprising variety of applications, eg. in path planning, computer vision and machine perception [19], meshing [6] etc. It is a widely studied subject, and several algorithms exist for their computation.

The VD of a given set of geometric objects (sites) in the plane is the partition of the plane into regions (cells), where each site S is associated to the region consisting of all points for which S is the nearest site, compared to any other site of the (so called) generating set.

If we take a set of points as generating set, and as distance between a site and an arbitrary point the length of their connecting line segment, we have the classic VD under the Euclidean metric. The distance induces a scalar *distance field*distance field over the plane for every site in the generating set.

There are at least two ways to generalize the construction of VD's, in order to meet applications' needs; to allow non-punctual sites, e.g. circles, under the Euclidean distance field, or to attach distance fields other than the Euclidean to the sites, for example the *anisotropic diagram* (see Section 5.1.2). These generalizations lead to curved VD's, having algebraic bisectors, see [16] for an expository paper. The Voronoi diagram can be represented as the projection of the distance field to the domain [31].

Nearest neighbor search for a query point q refers to determining which cell in the VD contains q. It is usual in applications to seek for the VD of a bounded planar domains, instead of considering the whole of $\mathbb{R}^2$. This computation is known as the *constrained VD*; the method presented here is indeed a local method for constrained VD's.

Our method computes Voronoi diagrams using the lower envelope of the distance fields. The latter exhibit an algebraic degree which is usually quite low, thus handling them in an algebraic fashion is computationally advantageous. The method is inspired by, and uses tools from the field of geometric modeling and CAGD. Polynomial curves constitute a major branch of the research in CAGD, and flexible curve representations have been developed in this frame. Therefore, we aim at bringing data representations and algorithmic experience from CAGD to this fundamental problem of computational geometry.

Algebraic curves are also main objects of study when looking at VD's and their

bisector sets. Subdivision techniques for meshing implicit curves [67] are quite spread in CAGD and provide fast and robust solutions in real-life applications. We are going to employ existing results [67, 70] on treating such curves in meshing VDs.

Our algorithm works directly on the distance field induced by the sites and does not require a representation of the sites themselves in the input. We use the method presented in Chapter 4 for the treatment of algebraic curves and their arrangements. The outpus is polygonal approximations of the Voronoi cells, and the method is applicable to all inputs where the distance field of every site can be expressed by a polynomial. This expression may be explicit, but also implicit. Section 5.1.2 presents a list of distance fields for commonly encountered VDs'.

Our implementation is done in C++, using double precision arithmetic. The algorithm is naturally parallelizable, since it applies locally and independently on different domains of the plane. We did not focus on insertion of new sites, or point queries, yet these operations can easily be added, without significant changes. On the other hand, we focus on genericy of the framework, the correctness of the result and the ability to treat arbitrary diagrams, if certain reasonable algorithmic prerequisites are met (cf. Section 5.2).

5.1.1 Some existing work

The bibliography on the subject is vast, and many alternative strategies have been proposed, but let us mention some of the latest developments.

The bibliography on VD's is vast, and many alternative strategies have been proposed, but let us mention some of the latest developments. Boada *et al.* [14] use a subdivision approach to compute approximate VD's. The closest site to the corners of a subdivided domain is computed and used to deduce the Voronoi cell in which the box belongs to. Their output is proved to converge to the exact VD, but the output is not always topologically correct, since corner signs alone are not in one-to-one correspondence with the possible configurations of bisectors inside rectangular domains.

Setter *et al.* [94] compute VD's using divide-and-conquer of lower envelopes and exact computations. They recursively build the diagram, by going down to two sites and then merging up a full VD. They provide good treatment of degenerate cases, and give a complexity analysis for randomized inputs. They comment that computation speed may be limited due to extensive use of symbolic computation.

Divide-and-conquer techniques are also utilized in Aichholzer *et al.* [1], to propose a VD algorithm for general shapes. They exploit connections to medial axis and use a plane-sweep technique. Their method avoids computing redundant pieces of bisectors that will then be rejected.

In Seong *et al.* [93], the VD of parametric NURBS-curves is explored. They

approximate bisectors as algebraic curves in parameter-space and use real solving to find their intersections globally, and then trim away the unwanted parts.

Emiris *et al.* [43] present an efficient, certified algorithm for VD's, specialized to certain families of closed planar curves, notably ellipses. They use adapted numerical techniques to reduce the degree of predicates to be evaluated. Their algorithm becomes exact if they choose to use a special iterated resultant for the "InCircle" predicate.

5.1.2 Voronoi diagrams and distance fields

In this section we recall a non-exclusive list of VD-types that we are interested in, and briefly introduce the main tools and representations that build up our algorithm.

A restriction of the input of our algorithm is that the distance field must be polynomial. Thus we must come up with transformed distance functions for a VD-type, before being able to compute it.

In the simple case of Euclidean VD the input is a list of (squared) distance algebraic functions $(f_1, \ldots, f_n)$. The lower envelope changes if we work with squared fields, yet the VD (that is, its projection) remains intact under squaring, or under any other invertible and strictly increasing transformation.

For a point $q = (x, y)$, the distance between q and the Voronoi site attached to f_i, *as viewed by* p_i is given by $\texttt{dist}_i(q) = f_i(q)$. Now one can define the Voronoi cell of $i-$th site as:

$$\begin{aligned} \mathrm{Vor}(p_i) &= \{q \in \mathbb{R}^2 \ : \ \texttt{dist}_i(q) \leq \texttt{dist}_j(q), \ j = 1 \ldots n\} \\ &= \{q \in \mathbb{R}^2 \ : \ \min_{1 \leq j \leq n} \{\texttt{dist}_j(q)\} = \texttt{dist}_i(q)\}. \end{aligned}$$

Therefore the VD can be retrieved as the the projection of the lower envelope of the distance fields [31], also known as a *minimization diagram*.

Different distance fields $\texttt{dist}_i(q), \ q \in \mathbb{R}^2$, give rise to different types of VD's, including:

- ◇ Euclidean VD of points $p_i = (x_i, y_i)$:
  $\texttt{dist}_i(x, y) = \|p - p_i\|^2 = (x - x_i)^2 + (y - y_i)^2$.

- ◇ VD of points $p_i = (x_i, y_i)$ under the ℓ_p-metric, p even:
  $\texttt{dist}_i(q) = (x - x_i)^p + (y - y_i)^p$.

- ◇ Anisotropic diagram of points $p_i = (x_i, y_i)$ with weights $w_i \in \mathbb{R}$ [61]:
  $\texttt{dist}_i(q) = (q - p_i)^T M_i (q - p_i) - w_i$,
  with $M_i \in \mathbb{R}^{2\times 2}$ symmetric positive definite matrix.

- ◇ Power (or Laguerre) diagram of points with weights: $\texttt{dist}_i(q) = \|p - p_i\|^2 - w_i^2$.
- ◇ Möbius diagram of points: $\texttt{dist}_i(q) = v_i\|p - p_i\|^2 - w_i$, with $v_i, w_i \in \mathbb{R}$.
- ◇ Apollonius (or additively weighted) diagram of disks [38] with centers p_i and radii w_i: $\texttt{dist}_i(q) = \|q - p_i\| - w_i$.
- ◇ Euclidean VD of ellipses [43], or Euclidean VD of general closed parametric curves.

An algebraic function over a bounded rectangular domain $\mathcal{D} = [a, b] \times [c, d]$ can be represented efficiently by its Bernstein coefficients over $\mathcal{D}$. It is a local description achieved by using tensor-product Bernstein representation over $\mathcal{D}$. This representation is computed by means of DeCasteljau's algorithm. We refer the reader bach to Section 4.2, for more on the subject.

We use $f_i|_{\mathcal{D}}$ for the restriction of $f_i(x, y)$ in $\mathcal{D}$, that is, the Bernstein representation of f_i over the domain $\mathcal{D}$. For instance, for the anisotropic diagram, the f_i's are conics, thus every $f_i|_{\mathcal{D}}$ is defined over any rectangular $\mathcal{D}$ by 9 coefficients.

Note that some VD types in the previous list do not have polynomial distance fields, nor do they admit squaring to eliminate radicals. In such cases, it may be possible to express the distance field implicitly. For instance, if we want to use our framework to compute the Apollonius diagram, i.e. the Euclidean VD of circles with centers p_i and radii w_i, we need to treat the corresponding distance function $\texttt{dist}_i(q) = \|q - p_i\| - w_i$, which may be transformed to the algebraic equation $(\texttt{dist}_i(q) + w_i)^2 = \|q - p_i\|^2$. Therefore, distance field values $z := \texttt{dist}_i(x, y)$ are given in *implicit form* by the cone

$$g_i(x, y, z) := (x - x_i)^2 + (y - y_i)^2 - (z + w_i)^2 = 0, \tag{5.1}$$

i.e., given (x_0, y_0), values of $\texttt{dist}_i(x_0, y_0)$ are computed by solving $g_i(x_0, y_0, z) = 0$ for the smallest root > 0 w.r.t. z. Also note that the zero locus of $g_i(x, y, r) = 0$ for $r \in \mathbb{R}_{>0}$ contains the $r-$offset of the site.

This tri-variate function can be represented similarly over a cuboid, by tensoring 3 Bernstein bases as in (4.1). In Section 5.3, we shall adopt our algorithm to use implicit representations.

5.2 The algorithm

Our algorithm consists of two phases: The subdivision phase (Algorithm 5.1), followed by the reconstruction phase (Algorithm 5.3).

Two algorithmic ingredients must be specified in order to apply the method on a specific VD type:

1. **Field bound:** A way to bound the value range of a distance field over a given domain.

2. **Bisector tracking:** A way to compute an approximation of the bisector(s) or vertices in a given domain.

These computations need to be carried out on axis-aligned boxes of the subdivision. Note that at this level we do not emphasize on efficiency, e.g. the bounds could be bad, or the approximation very loose. Nevertheless, it is expected that these computations converge to the actual distance value or bisector, when the size of the boxes becomes smaller.

The first box that is computed as soon as the algorithm is launched is the one corresponding to the initial domain $\mathcal{D}_0$. This initial box $B(\mathcal{D}_0)$ carries all the distance functions of the input, in Bernstein form.

In the subdivision phase we compute a graph of boxes that span the VD. To do so, the *field bound* is used in a filtering process (Algorithm 5.2) in order to exclude most boxes and reach down to boxes intersecting the VD. These boxes contain Bernstein representations of sites that contribute to the part of the diagram inside the box.

In the reconstruction phase the boxes of the subdivision are traversed and the part of the VD that intersects the box is meshed locally using the second ingredient. These bisector segments are stitched together to recover the full VD.

5.2.1 Subdivision phase

The idea of a space-subdivision scheme has benn used sever times in thepresent thesis: A big domain of interest is divided into smaller ones until some (fast computed) conditions are fulfilled. Then every small item is treated independently.

The representation (4.1) can be subdivided coordinate-wise using DeCasteljau's algorithm [44], having a quadratic time w.r.t. the degree of the polynomial, i.e. constant in our case. When a sub-cell is computed, the filter excludes sites, thus gradually keeping only the functions that participate locally in the lower envelope.

By signature of a box $\mathrm{sign}(B(\mathcal{D}))$ we mean the site labels that are present in that box, i.e. $\mathrm{sign}(B(\mathcal{D})) = \{i \ : \ f_i|_{\mathcal{D}} \in B(\mathcal{D})\}$.

The main loop in Algorithm 5.1 starts by filtering the box $B(\mathcal{D})$. This allows to remove large valued f_i's, i.e. those that correspond to sites that are "away" from the box. Then, we check if the remaining fields are at most three. In this case the box is not subdivided anymore. Otherwise the box is split in two sub-domains,

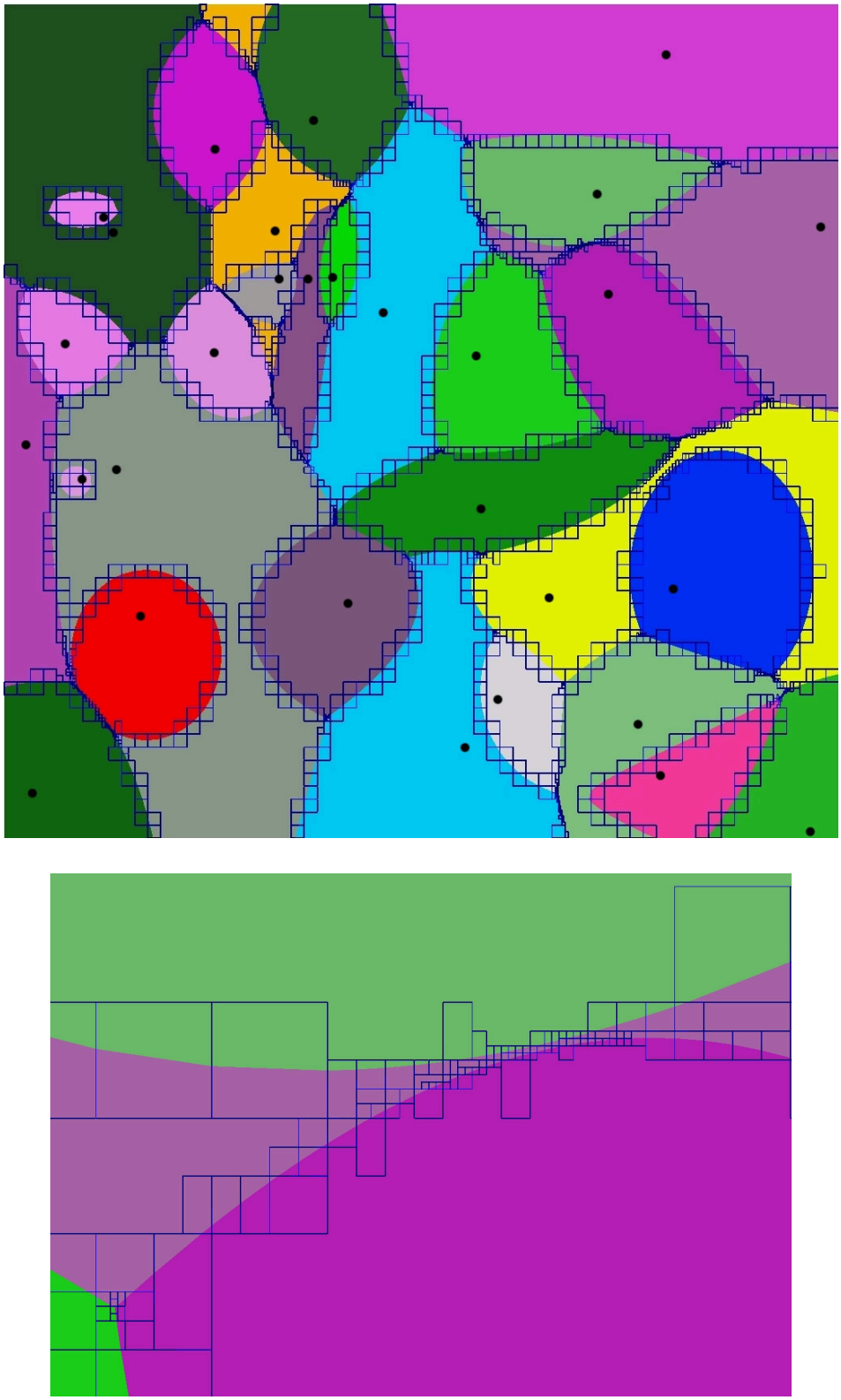

Figure 5.1: Up: Anisotropic diagram of 30 points over $[-3, 3]^2$. Note the existence of "widows" (cell components without points inside) and "islands" (cells surrounded entirely by another cell). Subdivision box span by Algorithm 5.1 is shown. Down: Detailed view at almost-touching bisectors.

Algorithm 5.1: Subdivision phase

Input: A set of distance fields $f_1, f_2, \ldots, f_n \in \mathbb{R}[x, y]$, a threshold $\varepsilon > 0$ and a rectangle domain $\mathcal{D}_0$.

Output: A graph G of boxes that span the minimization diagram of $f_1, \ldots, f_n$.

Compute $B(\mathcal{D}_0) := \{f_1|_{\mathcal{D}_0}, \ldots, f_n|_{\mathcal{D}_0}\}$;
Initialize stack Q and add $B(\mathcal{D}_0)$ on top of it;
Initialize empty box graph G;
while Q *is not empty* **do**
  Pop a box $B(\mathcal{D})$ from Q;
  Apply Algorithm 5.2 (filter) on $B(\mathcal{D})$;
  if $|B(\mathcal{D})| \leq 3$ *or* $|\mathcal{D}| < \varepsilon$ **then**
    Add $\tilde{B}(\mathcal{D}) := \{(f_i - f_j)|_{\mathcal{D}} \ : \ i < j \in \text{sign}(B(\mathcal{D}))\}$ to G ;
  else
    Split $B(\mathcal{D})$ into $B(\mathcal{D}_1)$ and $B(\mathcal{D}_2)$;
    Update adjacency graph G with $B(\mathcal{D}_1)$, $B(\mathcal{D}_2)$;
    Push $B(\mathcal{D}_1)$ and $B(\mathcal{D}_2)$ into Q ;
  end
end
return *G;*

along the longest of its sides. The new boxes are pushed in the stack to the loop continues.

If $B(\mathcal{D})$ is left with only one f_i then $\mathcal{D} \subseteq \text{Vor}(p_i)$, hence this box does not span the VD. If there are 2 active functions then the bisector locus $f_i - f_j = 0$ possibly intersects the box. If there are 3 or more functions then there probably exists a Voronoi vertex in $\mathcal{D}$. Boxes that contain degenerate Voronoi vertices, i.e. vertices that are equi-distant from more than 3 sites will always hold > 3 functions. Hence they will be subdivided until reaching threshold size $\varepsilon > 0$.

Whenever we reach a box holding at most three distance functions $f_1,\ f_2, f_3$, the box is *mutated*: This means that we form the differences $f_i|_{\mathcal{D}} - f_j|_{\mathcal{D}}$, $i < j$ and we store then in the box, in the place of $f_1|_{\mathcal{D}},\ f_2|_{\mathcal{D}},\ f_3|_{\mathcal{D}}$. Thus the box is transformed into *bisector box*. This box will be further treated in the next phase.

All boxes are either mutated and inserted in G or excluded (meaning that they are totally inside some Voronoi cell) after some time (depending on the size of the initial domain we compute) and the desired threshold. The subdivision stops when all cells are identified or if the threshold is reached. Cells that are not identified while reaching threshold size contain probably degenerate Voronoi vertices. Figure 5.1 shows the resulting box span for a set of anisotropic sites.

Three main operations constitute every iteration of subdivision: First, an appli-

cation of the filter of Section 5.2.2 on the subdivided boxes. Then, an execution of DeCasteljau's algorithm to split the representation of the distance functions. Finally, an update of the adjacency graph, i.e. connection of newly created boxes with their surrounding neighbors. These operations are $O(n)$ for a box with $|B(\mathcal{D})| = n$. The overall time of Algorithm 5.1 depends on the efficiency of the filter, since the fulfillment of the stopping condition $|B(\mathcal{D})| \leq 3$ depends on it.

5.2.2 Upper bounds and filtering

Not all bisectors are Voronoi edges. Furthermore, only a specific piece of the bisector curve of two sites is contained in the VD, the one connecting two adjacent Voronoi vertices. For instance, in the classic VD of points, only a linear amount (w.r.t. the number of sites) of line segments out of the totality of (line) bisectors contribute to the VD. This implies that, locally, only a few sites contribute to the VD and therefore the majority of sites should be filtered out. This idea leads to the filtering process described in this section.

The filter is based on the following fact: If the control grids of any two $f_i|_{\mathcal{D}}$, $f_j|_{\mathcal{D}}$ do not intersect, then the one that is superior to the other, say the $i-$th, does not contribute to the lower envelope over $\mathcal{D}$, or equivalently $\text{Vor}(i) \cap \mathcal{D} = \varnothing$. This is a direct consequence of the variation diminishing property of Bernstein representation [44].

Doing the previous test directly for a set of n sites requires $O(n^2)$ time to check all pairs. But we can do better: we compute an upper bound $U_{\mathcal{D}}$ on the lower envelope over $\mathcal{D}$, and then compare every control grid $f_i|_{\mathcal{D}}$ against this bound, excluding the functions that are found to be over it. This leads to a linear time filter, w.r.t. the number of grids $\mathcal{D}$.

Before computing $U_{\mathcal{D}}$, we need bounds on every $f_i|_{\mathcal{D}}$. For this we use the minimum (resp. maximum) Bernstein coefficient of $f_i|_{\mathcal{D}}$ to bound every f_i from below (resp. above). These extreme coefficients yield two supporting planes, parallel to $xy-$plane, that enclose the values $f_i(\mathcal{D})$. More sophisticated bounds exist, eg. [45, 86] and references therein. Computational time is proportional to the quality of the bounds. We choose to employ constant time bounds, given by minimum/maximum coefficients, since they are quite efficient in practice. Indeed, the control grid is known to converge with quadratic speed to the function it represents [89], thus a small number of refinements is needed to separate the control grids of two non-intersecting patches.

Now, to compute $U_{\mathcal{D}}$, an upper bound on the lower envelope over $\mathcal{D}$, consider $(x, y) \in \mathcal{D}$ and let $L(x, y)$ be the value of the lower envelope at $(x, y) \in \mathcal{D}$. We have:

$$L(x, y) = \min_i\{f_i(x, y)\} \leq \min_i\{\texttt{maxcoef}(f_i|_{\mathcal{D}})\}. \tag{5.2}$$

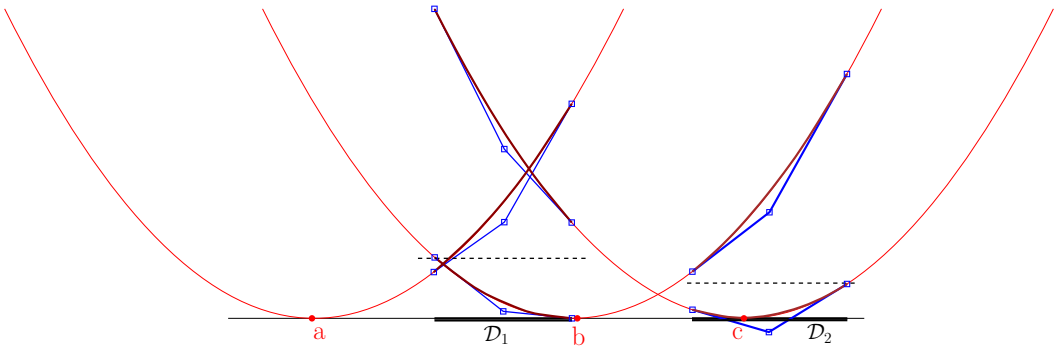

Figure 5.2: Filtering process applied to domains $\mathcal{D}_1$ and $\mathcal{D}_2$, in computing the VD of tree points $a, b, c \in \mathbb{R}$.

This directly proposes to take $U_{\mathcal{D}} = \min\{\texttt{maxcoef}(f_i|_{\mathcal{D}})\}$. Having $U_{\mathcal{D}}$, we can check which distance fields are bounded over this limit, and filter them out, see pseudo-code in Algorithm 5.2.

Figure 5.2 illustrates the filtering process (in one dimension lower for simplicity). Three point-sites a, b, c yield the red distance fields, that are squared Euclidean metrics. Domain $\mathcal{D}_1$ holds all three f_a, f_b, f_c (in blue), and the dotted line is the level of $U_{\mathcal{D}_1}$, equal to $\texttt{maxcoef}(f_b|_{\mathcal{D}_1})$. We see that $f_c|_{\mathcal{D}_1}$ is over the dotted line, thus it will be filtered out. Similarly, $\texttt{maxcoef}(f_c|_{\mathcal{D}_2})$ defines $U_{\mathcal{D}_2}$. Consequently, $f_b|_{\mathcal{D}_2}$ is excluded and $\mathcal{D}_2$ is dominated by site c, i.e. $\mathcal{D}_2 \subset \text{Vor}(c)$.

Algorithm 5.2: Filter sub-process

Input: A box $B(\mathcal{D}) = \{f_1|_{\mathcal{D}}, \ldots, f_s|_{\mathcal{D}}\}$.
Output: Filtered $\tilde{B}(\mathcal{D}) = \{f_{i_1}|_{\mathcal{D}}, \ldots, f_{i_k}|_{\mathcal{D}}\} \subseteq B(\mathcal{D})$.
$U_{\mathcal{D}} := \min\{\texttt{maxcoef}(f_i|_{\mathcal{D}}) \ : \ f_i|_{\mathcal{D}} \in B(\mathcal{D})\}$;
for $i = 1, \ldots, s$ **do**
  if $\texttt{mincoef}(f_i|_{\mathcal{D}}) \geq U_{\mathcal{D}}$ **then**
    $B(\mathcal{D}) := B(\mathcal{D}) \setminus \{f_i|_{\mathcal{D}}\}$;
  end
end
return $B(\mathcal{D})$;

5.2.3 Cell reconstruction phase

The box-graph computed by Algorithm 5.1 spans the VD, having edges between adjacent boxs. Traversing this graph, by navigating based on signature and sign of the bisector leads to the cells of the VD. In particular, we shall traverse all

boundary-contours of the Voronoi cells in counter-clock wise (CCW) order, completing open cells with pieces of the boundary of the initial domain $\mathcal{D}_0$.

As soon as the process starts, there is a pre-processing of the graph G. In this step, all the boxes of the graph G that touch the boundary of $\mathcal{D}_0$ are connected in a CCW-directed loop. This loop helps to constrain the reconstruction phase in $\mathcal{D}_0$. The boundary boxes are displayed in Figure 5.5 at the end of the this chapter.

First we discuss how every single box is treated, when encountered in the traversal. The simplest case of a box is one with one bisector $f_i - f_j = 0$. This equations expresses the locus of equi-distant points from sites i and j, and is typically approximated inside a domain $\mathcal{D}$ by a line segment connecting its intersections with $\partial\mathcal{D}$. But if its topology inside the box is not as simple, there may be several branches crossing $\mathcal{D}$. To include such cases, we resort to the methods of Chapter 3. that shall identify all different branches.

If there is more than one bisector that intersects $\mathcal{D}$, note that by definition of a Voronoi vertex, a candidate cell will contain $3, 6, \ldots, \binom{m}{2}$ active bisectors for a vertex of valency $3, 4, \ldots, m$ respectively. In this case we compute the arrangement (cf. Chapter 3) of the involved bisectors, including intersection points that are candidate Voronoi vertices. Computing Voronoi vertices is done by real solving.

The defining polynomial of a bisector $f_i - f_j$ is negative over the (open) Voronoi cell of i and positive over the Voronoi cell of j. The traversal of the spanning boxes uses this sign information: The signs of the bisector on the corners of $\mathcal{D}$ give us the correct orientation for tracking the cells coreesponding to i and j.

Algorithm 5.3 summarizes the reconstruction process. Starting from an arbitrary box, the boundary of the cell of site i is tracked by traversing the box-span G. The navigation is done based on the sign of the bisector $f_i - f_j$. When a vertex is reached, j is updated and the process continues until the recovery of the whole partition V.

The output of this algorithm is a polygonal approximation of each Voronoi cell.

Fig. 5.3 demonstrates bisector computation. The control polygon in $\mathcal{D}_1$ of the equation $f_a - f_b = 0$ is shown in green. Its intersection with the domain defines the bisector point (in 1D) of sites a, b.

5.3 The implicit method

In this section we sketch how the framework is extended to implicitly defined distance fields.

A restriction of the method presented in Section 5.2 is that the distance function must be polynomial. Thus we must come up with transformed distance functions for a VD type, before being able to compute it. But there are distance fields that cannot be made polynomial by a mere squaring, or by another suitable transforma-

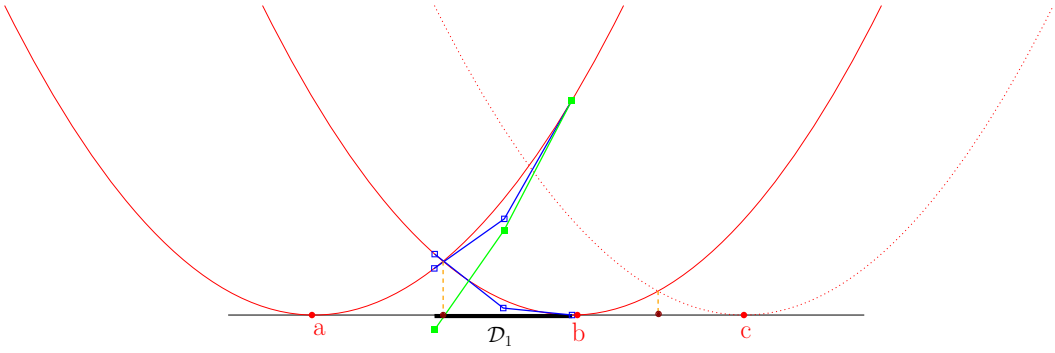

Figure 5.3: Approximating the bisector of a and b.

Algorithm 5.3: Cell reconstruction phase

Input: A graph G of boxes that span the minimization diagram of $f_1, \ldots, f_n$.
Output: A partition V of $\mathcal{D}_0$, defined by meshing the minimization diagram of $f_1, \ldots, f_n$.

Join border boxes of G in a CCW loop ;
Initialize $V[i] = \varnothing$, $i = 1, \ldots, n$;
foreach *node (box)* $B(\mathcal{D})$ *of* G **do**
  foreach *label* $i \in \text{sign}(B(\mathcal{D}))$ **do**
    foreach *label* $j \in \text{sign}(B(\mathcal{D})),\ j \neq i$ **do**
      • Walk in G, starting from $B(\mathcal{D})$, and track bisector (i, j) along boxes with label i ;
      • If some $B'(\mathcal{D})$ contains a vertex, set $j := j'$, $(i, j') \in B'(\mathcal{D})$ and continue, until $B(\mathcal{D})$ is reached for the second time ;
      • Add tracked cell component to $V[i]$.
    end
  end
end
return *V;*

tion.

A nice polynomial formula even for Euclidean distance fields no longer exists when the sites are not points. In such cases it is natural to represent the function in implicit form, that is, as a polynomial $g \in \mathbb{R}[x, y, z]$, s.t. a triple satisfying $g(x, y, z) = 0$, with $z > 0$ and minimal among the solutions implies that (x, y) is at distance z from the site.

Some diagrams that fall in this class are, as already mentioned, the Apollonius diagram, the VD of ellipses, or the VD of closed curves given by a support function representation [48]. We show how we can extract this implicit representation in 5.3.3.

In order to compute a representation of a tri-variate polynomial $g_i(x, y, z)$ over the initial domain $\mathcal{D}_0 \subset \mathbb{R}^2$, we must first compute an interval I_0 for the third variable. This must be big enough to contain the $z-$values of zeros of g. Therefore, we need initial upper bound for the roots of the univariate interval polynomial $g_i(\mathcal{D}, z)$. The latter is computed by interval arithmetic. After that, known univariate bounds, e.g. Cauchy's bound can be applied to get I_0. As a result, we feed the subdivision algorithm Algorithm 5.1 with $g_i|_{\mathcal{D}_0 \times I_0}$, for every site i in the generating set. Note that this box shall be split only w.r.t. x or y.

The same ingredients apply, i.e. the "field bound" and "bisector tracking" (Section 5.2) are needed, in order to run the scheme. The subdivision and reconstruction is done the same way, on 3D boxes that enclose implicit bisector branches.

5.3.1 Field bound

For $g|_{\mathcal{D}\times I} = \sum_{i,j,k=0}^{d_x,d_y,d_z} \gamma_{ijk} B^i_{d_x}(x)\, B^j_{d_y}(y)\, B^k_{d_z}(z)$ we set $m_k = \min_{i,j}\{\gamma_{ijk}\}$ and $M_k = \max_{i,j}\{\gamma_{ijk}\}$. Let , as in Section 2.3,

$$m_g(z) := \sum_{k=0}^{d_z} m_k\, B^k_{d_z}(z) \qquad , \qquad M_g(z) := \sum_{k=0}^{d_z} M_k\, B^k_{d_z}(z). \tag{5.3}$$

These polynomials are defined over the $z-$interval I and enclose the range of $z-$values, since, for $(x, y, z) \in \mathcal{D} \times I$,

$$g(x, y, z) \geq \sum_{k=0}^{d_z} m_k B^k_{d_z}(z) \sum_{j=0}^{d_x} B^i_{d_x}(x) \sum_{j=0}^{d_y} B^j_{d_y}(y) \; = m_g(z),$$

and similarly for $M_g(z)$. Consequently, a lower bound on $z-$values is given by:

$$\mu_I := \begin{cases} \text{first root of } M_g(z) \text{ in } I & \text{if } M_0 < 0 \\ \text{first root of } m_g(z) \text{ in } I & \text{if } m_0 > 0 \\ 0 & \text{otherwise} \end{cases},$$

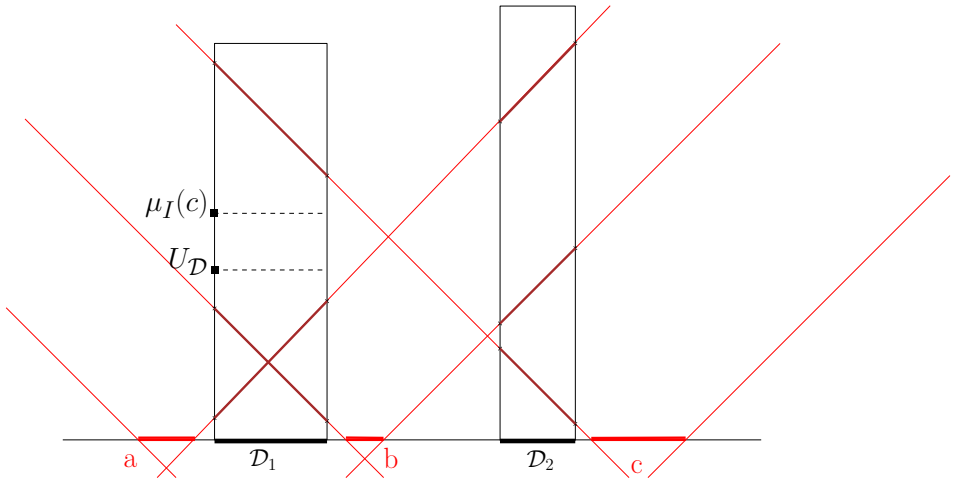

Figure 5.4: Instance of Apollonius diagram in 1D: sites a, b, c are line segments and distance fields are bi-variate (implicit) cones.

and similarly for the upper bound $\mathcal{M}_I$, using the last real roots of $m_g(z)$ or $M_g(z)$ in I. Finally, using these enclosures we get $U_{\mathcal{D}} = \min\{\mathcal{M}_I(i) \ : \ g_i \in \mathcal{D}\}$, and Algorithm 5.2 applies.

5.3.2 Bisector tracking

Bisectors are defined by the (projection of the) intersection of two implicit surfaces, thus they are implicit spatial curves. One way to get the projection is using resultants, but this is would be costly to do in every box, and would increase the degree of the polynomial to handle. Isotopic meshing of 3D curves, given in Bernstein form, has been implemented in [67], and yields a suitable way to derive the projection. Figure 5.4 shows a snapshot of the implicit method. Sites are segments on the real axis and distance fields are implicit cones. The box $\mathcal{D}_1 \times I_1$ contains 3 bi-variate Bernstein polynomials. Based on the computed $U_{\mathcal{D}_1}$ and the lower bound $\mu_{I_1}(c)$, the $f_c|_{\mathcal{D}_1}$ is filtered out.

5.3.3 Equations for implicit distance fields

The offset of a site is closely related to the distance function. The $z-$offset is the set of points at distance z from the site. Therefore, we need to compute an implicit equation of the $z-$offset, having z as a parameter.

For the Apollonius diagram, the $z - offset$ of a circle-site centered at p_i and radius w_i is the circle of the same center and radius $z + w_i$ (see equation 5.1).

If the site in question is a closed curve given by support function parametrization (cf. [48]) $h(t)$, $t \in \mathbb{S}^1$, then it's $z-$offset is given by $h(t) + z$, in support form.

diagram / # sites		50	100	200	400	800
Euclidean	time	0.85	1.9	4.8	10.8	30.7
	boxes	1959	3136	5351	8322	14616
Anisotropic	time	1.0	2.9	4.7	11.6	31.5
	boxes	2269	3886	5214	8552	14705

Table 5.1: Execution details for random inputs in $[-2, 2]^2$.

Let $n(t) : \mathbb{R} \to \mathbb{S}^1$ be a parametrization of the circle $\mathbb{S}^1$. The parametrization of the curve is over the unit circle, and associates to every point $n(t)$ of the unit circle the curve-point that has normal vector $n(t)$. Applying implicitization, e.g. eliminating t from $(h(t) + z) \cdot n(t) - (x, y) = 0$ we arrive to the implicit form of the $z-$offset.

Anton *et al.* [4] compute, using resultants, the offset curve of radius z of a conic, eg. an ellipse $\mathcal{E}$. This is a polynomial of degree 4 w.r.t. z and of degree 8 w.r.t. x, y. For any specific point $q = (x, y)$ outside of the ellipse, we have an irreducible univariate polynomial that has exactly one real positive solution, corresponding to $\texttt{dist}(q, \mathcal{E})$. We note that the same equation can be derived if we consider a support function parametrization of the ellipse.

5.4 Experimentation

We run experiments on Euclidean and anisotropic VD. We used a Fedora 15 machine with an Intel Xeon CPU and 4GB of memory. For these tests, we set $\varepsilon = 0.001$ and $\mathcal{D}_0 = [-2, 2]^2$. Also, for the sake of a smooth result, we set a maximum size of boxes, i.e. all boxes are subdivided until their longest side is at most $\varepsilon = 0.05$. Table 5.1 reports execution time (in secs) and number of boxes produced, for random inputs of 50 up to 800 sites in $\mathcal{D}_0$.

The timings are of the same order for both VD types. This can be explained by the fact that the process discards the linear nature of the Euclidean VD. On the other hand, it is applied to the non-linear, anisotropic case, where bisectors are general conics, with the same performance. This adaptability to general topologies, without an extra computational effort, seems to be an advantage of our approach.

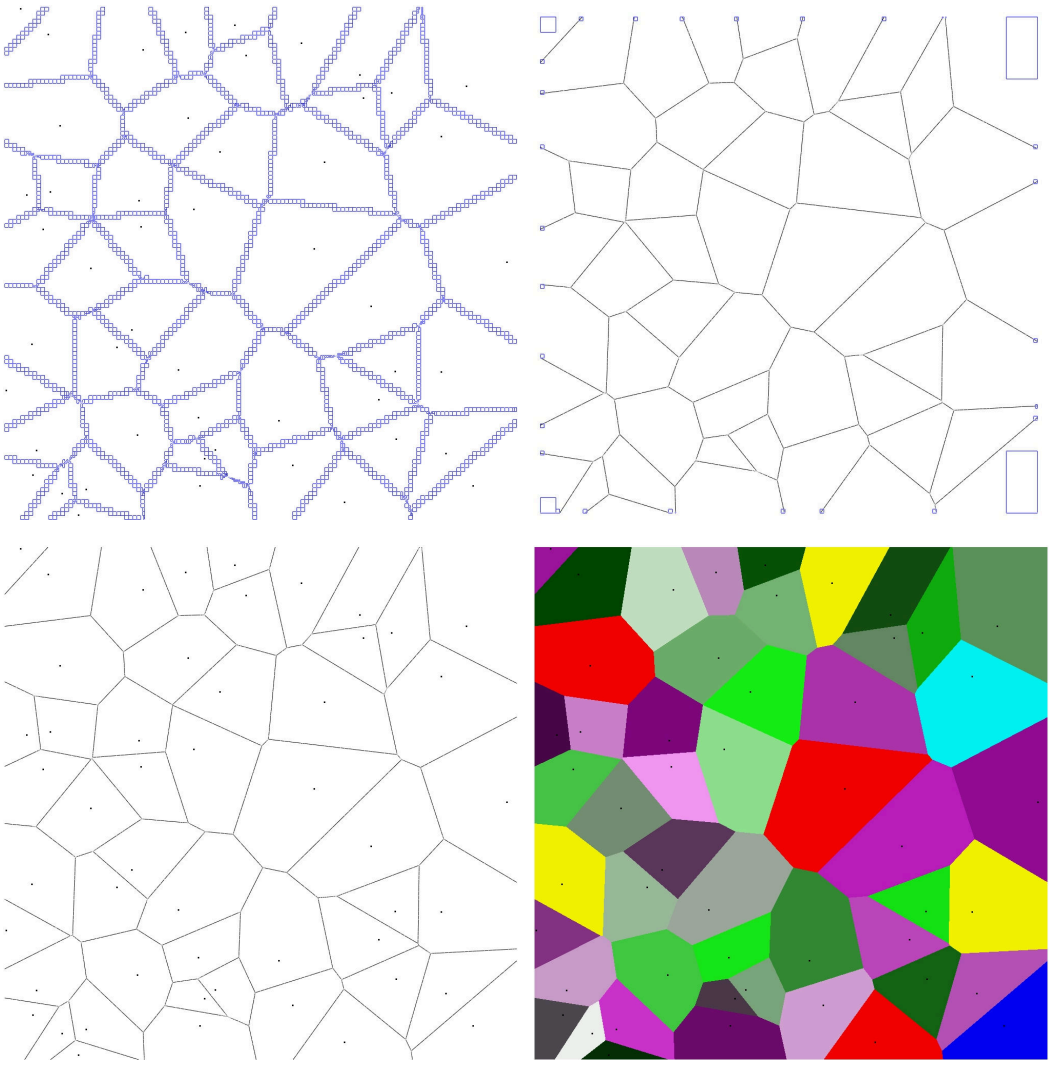

Figure 5.5: Euclidean VD of points over $[-2, 2^2]$. Up left: Subdivision span. Up right: Bisectors and boundary boxes. Down left: Bisectors are meshed. Down right: Voronoi cells are reconstruced.

Conclusion and outlook

CAD applications rely on fast numeric computation of real points that typically appear as solutions of polynomial systems. The use of robust subdivision algorithms for this task seems indispensable due to their performance and practical behavior. In this class of algorithms, we contributed the generalization of the classic *continued fraction algorithm* to multivariate systems. We also provided bounds on its time complexity as well as practical efficiency results.

A standing challenge in computing approximate algebraic points is singular point identification and treatment. We contributed an adaptive method for this problem, based on the dual structure of isolated points and on interval arithmetic. Under certain conditions, it allows certification of a singular point in a given rectangular domain, even for *approximate input data*, commonly encountered in CAGD.

We applied subdivision techniques to the treatment of the connected components of a semi-algebraic set. We deduced and implemented a new algorithm, based on a numerically stable scheme, to compute *certified* polygonal regions for the connected components of a planar semi-algebraic set.

Finally, we formulated the problem of computing Voronoï diagrams in an algebraic setting, which allows the description of a variety of different diagrams. A symbolic-numeric bisection algorithm following this formulation was developed, implemented and tested.

Reliable computing calls for new algorithms but also robust implementations. Towards this long-term goal, one faces the old Pythagoras' dilemma: *Compute symbolically or approximate?* Against this question, our general conclusion is to pursue *approximate yet certified computing*, exploiting the algebraic and geometric structure of the problem at hand.

We demonstrated cases of using effective algebraic methods on critical geometric operations: intersection or self-intersections of curves, topology analysis, arrangement computation of semi-algebraic sets, Voronoï diagrams. Undoubtedly, there is room for further investigation in the aforementioned problems:

◇ New criteria for use in subdivision solvers need to be developed, towards improving even more their complexity and practical behavior. Singularity treatment and multiple point deflation are key ingredients to gain in robustness.

◇ Methods and algorithms tipped herein for topology analysis or arrangement computation should be extended to 3D-shapes (probably starting with the special case of *arrangements of quadrics*), or to higher-dimensional manifolds.

◇ The problem of solving geometric constraints that involve inequalities, formulated as a semi-algebraic system, now seems tractable from the point of view of subdivision-based methods.

Les applications en CAO reposent sur le calcul rapide et numérique des points réels, qui apparaissent généralement comme solutions de systèmes polynomiaux. L'utilisation d'algorithmes de sous-division robustes pour cette tâche semble indispensable en raison de leur performance et de leur comportement pratique. Dans cette classe d'algorithmes, nous avons généralisé *l'algorithme de fractions continues* classique aux systèmes à plusieurs variables. Nous avons montré des bornes de complexité en temps ainsi que des résultats d'efficacité pratique.

Un défi permanent dans le calcul approché de points algébriques est l'identification et le traitement des points singuliers. Nous avons apporté une méthode adaptative, basé sur la structure duale des points isolés et sur l'arithmétique d'intervalles. Sous certaines conditions, cela permet la certification d'un point singulier dans un domaine rectangulaire donné, même pour *des données d'entrée approchées*, communément rencontrées en CAGD.

Nous avons appliqué des méthodes de sous-division au traitement des composantes connexes d'un ensemble semi-algebrique. Nous avons déduit et implanté un nouvel algorithme, basé sur une méthode numérique stable, pour calculer des régions polygonales *certifiés* pour toutes les composantes connexes d'un ensemble semi-algébrique dans le plan.

Enfin, nous avons formalisé le problème du calcul de diagrammes de Voronoï par enveloppes inférieures, dans un cadre algébrique, qui permet la description d'une variété de diagrammes différents. Un algorithme symbolique-numérique de dichotomie suivant cette formalisation a été développé, implémenté et testé.

Le calcul fiable nécessite de nouveaux algorithmes, mais aussi des implémentations robustes. On est alors confronté au Dilemme de Pythagore: *Calculer symboliquement ou approcher?* Notre conclusion générale est de proposer des calculs *approchés mais certifiées*, exploitant la structure algébrique et géométrique des problèmes traités.

Nous avons démontré des cas d'utilisation effective des méthodes algébriques sur des opérations géométriques critiques: l'intersection ou l'auto-intersections de courbes, l'analyse de topologie, le calcul d'arrangements des ensembles semi-algébriques, des diagrammes de Voronoï. Des nombreuses pistes restent à explorer:

◇ Des nouveaux critères pour les solveurs de sous-division doivent être développés, pour l'amélioration de leur complexité et leur comportement pratique. Le traitement de cas singuliers et la déflation des points multiples sont les ingrédients clés pour gagner en robustesse.

◇ Les algorithmes présentés ici pour l'analyse topologique ou le calcul d'arrangements devraient être étendu à la 3D (probablement en commencant par le cas particulier des *arrangements de quadriques*), ou même aux dimensions supérieures.

◇ Le problème de la résolution de contraintes géométriques qui impliquent des inégalités, formulé comme un système semi-algébrique, pourrait desormais être traité par des méthodes de sous-division.

Bibliography

[1] O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, E. Pilgerstorfer, and M. Rabl. Divide-and-conquer algorithms for Voronoi diagrams revisited. *Comput. Geom. Theory Appl.*, 43:688–699, 2010. (cited on page 89)

[2] L. Alberti, B. Mourrain, and J. Wintz. Topology and arrangement computation of semi-algebraic planar curves. *Comput. Aided Geom. Des.*, 25:631–651, November 2008. (cited on pages 40, 63 and 76)

[3] A. Alesina and M. Galuzzi. A new proof of Vincent's theorem. *L'Enseignement Mathémathique*, 44:219–256, 1998. (cited on page 19)

[4] F. Anton, I. Emiris, B. Mourrain, and M. Teillaud. The offset to an algebraic curve and an application to conics. In O. Gervasi, M. Gavrilova, V. Kumar, A. Laganà, H. Lee, Y. Mun, D. Taniar, and C. Tan, editors, *Computational Science and Its Applications – ICCSA 2005*, volume 3480 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin / Heidelberg, 2005. (cited on page 101)

[5] M. Atiyah and I. MacDonald. *Introduction to Commutative Algebra*. Addison-Wesley, 1969. (cited on page 43)

[6] C. Bajaj. A Laguerre voronoi based scheme for meshing particle systems. *Japan Journal of Industrial and Applied Mathematics*, 22:167–177, 2005. (cited on page 88)

[7] C. L. Bajaj and G. Xu. Regular algebraic curve segments (iii)–applications in interactive design and data fitting. *Computer Aided Geometric Design*, 18(3):149 – 173, 2001. (cited on page 76)

[8] M. Bartoň and B. Jüttler. Computing roots of polynomials by quadratic clipping. *Comp. Aided Geom. Design*, 24:125–141, 2007. (cited on page 6)

[9] S. Basu, R. Pollack, and M.-F. Roy. Complexity of computing semi-algebraic descriptions of the connected components of a semi-algebraic set. In *ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 25–29, New York, NY, USA, 1998. ACM. (cited on page 65)

[10] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, Berlin, 2003. ISBN 3-540-00973-6. (cited on pages 64 and 65)

[11] S. Béla. *Fat Arcs and Fat Spheres for Approximating Algebraic Curves and for Solving Polynomial Systems*. PhD thesis, Johannes Kepler University, 2011. (cited on pages 2 and 7)

[12] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980. (cited on page 69)

[13] L. Blum, F. Cucker, M. Shub, and S. Smale. Complexity and real computation: a manifesto. *Internat. J. Bifur. Chaos Appl. Sci. Engrg.*, 6(1):3–26, 1996. (cited on pages 24 and 31)

[14] I. Boada, N. Coll, N. Madern, and J. Antoni Sellares. Approximations of 2d and 3d generalized voronoi diagrams. *Int. J. Comput. Math.*, 85(7):1003–1022, 2008. (cited on page 89)

[15] J. Bochnak, M. Coste, and M.-F. Roy. *Géométrie Algébrique Réelle*. Springer-Verlag, Heidelberg, 1987. (cited on page 64)

[16] J.-D. Boissonnat, C. Wormser, and M. Yvinec. Curved voronoi diagrams. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 67–116. Springer Berlin Heidelberg, 2006. (cited on page 88)

[17] E. Bombieri and A. van der Poorten. Continued fractions of algebraic numbers. In *Computational algebra and number theory (Sydney, 1992)*, pages 137–152. Kluwer Acad. Publ., Dordrecht, 1995. (cited on pages 25 and 26)

[18] L. Busé. *Étude du résultant sur une variété algébrique*. These, Université de Nice Sophia-Antipolis, Dec. 2001. (cited on page 7)

[19] S.-W. Cheng, H.-S. Na, A. Vigneron, and Y. Wang. Querying approximate shortest paths in anisotropic regions. In *Proceedings of the twenty-third annual symposium on Computational geometry*, SCG '07, pages 84–91, New York, NY, USA, 2007. ACM. (cited on page 88)

[20] G. Chèze, J.-C. Yakoubsohn, A. Galligo, and B. Mourrain. Computing nearest gcd with certification. In *Symbolic-Numeric Computation (SNC'09)*, pages 29–34, Japon Kyoto, 2009-08-27. ACM New York, NY, USA. (cited on page 30)

[21] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes Comput. Sci.*, pages 134–183. Springer-Verlag, 1975. (cited on page 64)

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001. (cited on page 67)

[23] M. Coste. An introduction to semi-algebraic geometry. RAAG network school, 2002. (cited on page 64)

[24] F. Cucker, T. Krick, G. Malajovich, and M. Wschebor. A numerical algorithm for zero counting, I: Complexity and accuracy. *J. Complex.*, 24(5-6):582–605, 2008. (cited on page 30)

[25] F. Cucker, T. Krick, G. Malajovich, and M. Wschebor. A numerical algorithm for zero counting, II: Distance to ill-posedness and smoothed analysis. *Journal of Fixed Point Theory and Applications*, 10.1007/s11784-009-0127-4, 2009. (cited on pages 30 and 31)

[26] B. H. Dayton and Z. Zeng. Computing the multiplicity structure in solving polynomial systems. In *ISSAC '05: Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 116–123, New York, NY, USA, 2005. ACM. (cited on pages 40, 42, 47, 54, 57 and 59)

[27] J. Dedieu and J. Yakoubsohn. Computing the real roots of a polynomial by the exclusion algorithm. *Numerical Algorithms*, 4(1):1–24, 1993. (cited on page 7)

[28] D. Diatta, Niang. *Calcul effectif de la topologie de courbes et surfaces algébriques réelles*. These, Université de Limoges, Sept. 2009. (cited on page 64)

[29] A. Dickenstein and I. Z. Emiris. Multihomogeneous resultant formulae by means of complexes. *J. Symb. Comput.*, 36:317–342, September 2003. (cited on page 7)

[30] M. S. E. Din and L. Zhi. Computing rational points in convex semialgebraic sets and sum of squares decompositions. *SIAM Journal on Optimization*, 20(6):2876–2889, 2010. (cited on page 64)

[31] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, 1:25–44, 1986. 10.1007/BF02187681. (cited on pages 88 and 90)

[32] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the Descartes method. In *ISSAC 2006*, pages 71–78. ACM, New York, 2006. (cited on page 6)

[33] D. Eisenbud and H. Levine. An algebraic formula for the degree of a c^∞ map germ. *The Annals of Mathematics*, 106(1):pp. 19–44, 1977. (cited on page 56)

[34] G. Elber and M.-S. Kim. Geometric constraint solver using multivariate rational spline functions. In *Proc. of 6th ACM Symposium on Solid Modelling and Applications*, pages 1–10. ACM Press, 2001. (cited on page 7)

[35] M. Elkadi and B. Mourrain. *Introduction à la résolution des systèmes d'équations algébriques*, volume 59 of *Mathématiques et Applications*. Springer-Verlag, 2007. (cited on pages 45 and 56)

[36] I. Emiris, A. Mantzaflaris, and B. Mourrain. Yet another algorithm for generalized voronoi diagrams. Submitted, 2011. (cited on page 87)

[37] I. . Z. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, Computer Science Division, Univ. of California at Berkeley, Dec. 1994. (cited on page 7)

[38] I. Z. Emiris and M. I. Karavelas. The predicates of the apollonius diagram: Algorithmic analysis and implementation. *Comput. Geom. Theory Appl.*, 33:18–57, January 2006. (cited on page 91)

[39] I. Z. Emiris and A. Mantzaflaris. Multihomogeneous resultant formulae for systems with scaled support. In *ISSAC '09: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, pages 143–150, New York, NY, USA, 2009. ACM. (cited on page 7)

[40] I. Z. Emiris and B. Mourrain. Matrices in elimination theory,. *Journal of Symbolic Computation*, 28(1-2):3 – 43, 1999. (cited on page 7)

[41] 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*, volume 5045 of *LNCS*, pages 57–82. Springer Verlag, 2008. (cited on page 6)

[42] I. Z. Emiris, B. Mourrain, and E. P. Tsigaridas. The DMM bound: Multivariate (aggregate) separation bounds. In S. Watt, editor, *Proc. 35th ACM Int'l Symp. on Symbolic & Algebraic Comp. (ISSAC)*, pages 243–250, Munich, Germany, July 2010. ACM. (cited on pages 28, 29 and 34)

[43] I. Z. Emiris, E. P. Tsigaridas, and G. M. Tzoumas. Exact delaunay graph of smooth convex pseudo-circles: general predicates, and implementation for ellipses. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 211–222, New York, NY, USA, 2009. ACM. (cited on pages 81, 90 and 91)

[44] G. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. (cited on pages 66, 69, 92 and 95)

[45] J. Garloff and A. Smith. A comparison of methods for the computation of affine lower bound functions for polynomials. In C. Jermann, A. Neumaier, and D. Sam, editors, *Global Optimization and Constraint Satisfaction*, volume 3478 of *Lecture Notes in Computer Science*, pages 364–364. Springer Berlin / Heidelberg, 2005. (cited on pages 6 and 95)

[46] J. Garloff and A. P. Smith. Investigation of a subdivision based algorithm for solving systems of polynomial equations. *Journal of Nonlinear Analysis*, 47(1):167–178, 2001. (cited on page 23)

[47] M. Giusti, G. Lecerf, B. Salvy, and J.-C. Yakoubsohn. On location and approximation of clusters of zeros: Case of embedding dimension one. *Foundations of Computational Mathematics*, 7:1–58, 2007. 10.1007/s10208-004-0159-5. (cited on pages 40 and 55)

[48] J. Gravesen, Z. Šír, and B. Jüttler. Curves and surfaces represented by polynomial support functions. *Theoretical Computer Science*, 392:141–157, 2008. (cited on pages 99 and 100)

[49] R. M. Hardt. Triangulation of subanalytic sets and proper light subanalytic maps. *Invent. Math.*, 38(3):207–217, 1976/77. (cited on page 64)

[50] M. Hemmer, E. P. Tsigaridas, Z. Zafeirakopoulos, I. Z. Emiris, M. I. Karavelas, and B. Mourrain. Experimental evaluation and cross-benchmarking of univariate real solvers. In *Proc. 3rd ACM Int'l Work. Symbolic Numeric Computation (SNC)*, pages 45–54, New York, NY, USA, 2009. ACM. (cited on page 6)

[51] D. Henrion, J.-B. Lasserre, and J. Lofberg. GloptiPoly 3: moments, optimization and semidefinite programming. *Optimization Methods and Software*, 24(4-5):pp. 761–779, 08 2009. Rapport LAAS n° 07536 90C22; 47A57. (cited on page 64)

[52] H. Hironaka. Triangulations of algebraic sets. In *Algebraic geometry (Proc. Sympos. Pure Math., Vol. 29, Humboldt State Univ., Arcata, Calif., 1974)*, pages 165–185. Amer. Math. Soc., Providence, R.I., 1975. (cited on page 64)

[53] M. Hodorog, B. Mourrain, and J. Schicho. GENOM3CK - A Library for Genus Computation of Plane Complex Algebraic Curves Using Knot Theory, December 2010. ACM SIGSAM Communications in Computer Algebra, vol. 44, issue 174, pp. 198-200, ISSN:1932-2240. (cited on page 55)

[54] M. Hodorog, B. Mourrain, and J. Schicho. An Adapted Version of the Bentley-Ottmann Algorithm for Invariants of Plane Curve Singularities. In B. M. et al., editor, *Proceedings of the 11th International Conference on Computational Science and Its Applications, Part III, Session: Computational Geometry and Applications*, Lecture Notes in Computer Science, pages 121–131. Springer, Heidelberg, 2011. (cited on page 55)

[55] M. Hodorog and J. Schicho. A Regularization Method for Computing Approximate Invariants of Plane Curves Singularities. In L. Z. et al., editor, *Proceedings of the 4th International Workshop on Symbolic-Numeric Computation*. ACM, 2011. (cited on page 55)

[56] W. Kahan. A more complete interval arithmetic. *Lecture notes for a summer course at the University of Michigan*, 1968. (cited on page 55)

[57] G. N. Khimšiašvili. The local degree of a smooth mapping. *Sakharth. SSR Mecn. Akad. Moambe*, 85(2):309–312, 1977. (cited on page 77)

[58] A. Khintchine. *Continued Fractions*. University of Chicago Press, Chicago, 1964. (cited on page 26)

[59] C. Konaxis. *Algebraic algorithms for polynomial system solving and applications*. PhD thesis, National & Kapodistrian Univ. Athens, June 2010. In Greek. (cited on page 7)

[60] R. Krawczyk. Newton-algorithmen zur bestimmung von nullstellen mit fehlerschranken. *Computing*, 4(3):187–201, 1969. (cited on page 55)

[61] F. Labelle and J. R. Shewchuk. Anisotropic voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 191–200, New York, NY, USA, 2003. ACM. (cited on page 90)

[62] J. B. Lasserre. *Moments, Positive Polynomials and their Applications*, volume 1 of *Optimization Series*. Imperial College Press, 2009. (cited on page 64)

[63] G. Lecerf. Quadratic newton iteration for systems with multiplicity. *Foundations of Computational Mathematics*, 2:247–293, 2002. (cited on pages 40 and 57)

[64] P. Lévy. Sur les lois de probabilitié dont dependent les quotients complets et incomplets d' une fraction continue. *Bull. Soc. Math.*, 57:178–194, 1929. (cited on page 26)

[65] A. Leykin, J. Verschelde, and Z. A. Newton's method with deflation for isolated singularities of polynomial systems. *Theoretical Computer Science*, 359(1-3):111 – 122, 2006. (cited on pages 40, 42 and 54)

[66] A. Leykin, J. Verschelde, and A. Zhao. Higher-order deflation for polynomial systems with isolated singular solutions. In A. Dickenstein, F.-O. Schreyer, and A. Sommese, editors, *Algorithms in Algebraic Geometry*, volume 146 of *The IMA Volumes in Mathematics and its Applications*, pages 79–97. Springer New York, 2008. (cited on pages 40, 42, 45, 57 and 61)

[67] C. Liang, B. Mourrain, and J.-P. Pavone. Subdivision methods for the topology of 2d and 3d implicit curves. In B. Jüttler and R. Piene, editors, *Geometric Modeling and Algebraic Geometry*, pages 199–214. Springer Berlin Heidelberg, 2008. (cited on pages 89 and 100)

[68] T. Luu Ba. *Représentation matricielle implicite de coubres et surface algériques et applications*. PhD thesis, Université de Nice Sophia-Antipolis, July 2011. (cited on page 64)

[69] F. Macaulay. *The algebraic theory of modular systems*. Cambridge Univ. Press, 1916. (cited on pages 41, 42, 46 and 47)

[70] A. Mantzaflaris and B. Mourrain. A subdivision approach to planar semi-algebraic sets. In *Advances in Geometric Modeling and Processing*, volume 6130 of *Lecture Notes in Computer Science*, pages 104–123. Springer Berlin / Heidelberg, 2010. (cited on pages 63 and 89)

[71] A. Mantzaflaris and B. Mourrain. Deflation and certified isolation of singular zeros of polynomial systems. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation*, ISSAC '11, pages 249–256, New York, NY, USA, 2011. ACM. (cited on page 39)

[72] A. Mantzaflaris, B. Mourrain, and E. Tsigaridas. Continued fraction expansion of real roots of polynomial systems. In *SNC '09: Proceedings of the 2009 Conference on Symbolic Numeric Computation*, pages 85–94, New York, NY, USA, 2009. ACM. (cited on pages 6 and 55)

[73] A. Mantzaflaris, B. Mourrain, and E. Tsigaridas. On continued fraction expansion of real roots of polynomial systems, complexity and condition numbers. *Theoretical Computer Science*, 412(22):2312 – 2330, 2011. (cited on page 6)

[74] M. Marden. *Geometry of Polynomials*. American Mathematical Society, Providence, RI, 1966. (cited on page 19)

[75] M. G. Marinari, T. Mora, and H. Möller. Gröbner duality and multiplicities in polynomial system solving. In *Proceedings of the 1995 international symposium on Symbolic and algebraic computation*, ISSAC '95, pages 167–179, New York, NY, USA, 1995. ACM. (cited on pages 41 and 46)

[76] K. Mehlhorn and S. Ray. Faster algorithms for computing Hong's bound on absolute positiveness. *J. Symbolic Computation*, 45(6):677 – 683, 2010. (cited on page 6)

[77] R. Moore. A test for existence of solutions to nonlinear systems. *SIAM Journal on Numerical Analysis*, pages 611–615, 1977. (cited on page 22)

[78] B. Mourrain. *Approche effective de la théorie des invariants des groupes classiques*. PhD thesis, Centre de Mathématiques de l'École Polytechnique, Sept. 1991. (cited on page 40)

[79] B. Mourrain. Isolated points, duality and residues. *Journal of Pure and Applied Algebra*, 117-118:469 – 493, 1997. (cited on pages 41, 42, 44, 45, 46 and 47)

[80] B. Mourrain and J. Pavone. Subdivision methods for solving polynomial equations. *Journal of Symbolic Computation*, 44(3):292 – 306, 2009. Polynomial System Solving in honor of Daniel Lazard. (cited on pages 6, 7, 16, 30, 35, 36, 55 and 76)

[81] B. Mourrain, F. Rouillier, and M.-F. Roy. *Bernstein's basis and real root isolation*, pages 459–478. Mathematical Sciences Research Institute Publications. Cambridge University Press, 2005. (cited on page 6)

[82] T. Ojika, S. Watanabe, and T. Mitsui. Deflation algorithm for the multiple roots of a system of nonlinear equations. *Journal of Mathematical Analysis and Applications*, 96(2):463 – 479, 1983. (cited on page 40)

[83] V. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Rev.*, 39(2):187–220, 1997. (cited on page 6)

[84] V. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *J. Symbolic Computation*, 33(5):701–733, 2002. (cited on page 6)

[85] J. Pavone. *Auto-intersection de surfaces pamatrées réelles*. PhD thesis, Université de Nice Sophia-Antipolis, 2004. (cited on pages 2 and 7)

[86] J. Peters and X. Wu. Sleves for planar spline curves. *Computer Aided Geometric Design*, 21(6):615 – 635, 2004. (cited on pages 6 and 95)

[87] S. Pope and A. Szanto. Nearest multivariate system with given root multiplicities. *Journal of Symbolic Computation*, 44(6):606 – 625, 2009. (cited on page 41)

[88] A. Poteaux. *Calcul de développements de Puiseux et application au calcul du groupe de monodrmie d'une courbe algébrique plane*. PhD thesis, Université de Limoges, 2008. (cited on page 55)

[89] U. Reif. Best bounds on the approximation of polynomials and splines by their control structure. *Comput. Aided Geom. Des.*, 17(6):579–589, 2000. (cited on pages 6 and 95)

[90] S. Rump and S. Graillat. Verified error bounds for multiple roots of systems of nonlinear equations. *Numerical Algorithms*, 54:359–377, 2010. 10.1007/s11075-009-9339-3. (cited on pages 40, 41, 42, 43, 55 and 59)

[91] M. Safey El Din. *Resolution reelle des systemes polynomiaux en dimension positive*. PhD thesis, Pierre et Marie Curie University, 2001. (cited on page 7)

[92] M. Safey El Din. Testing sign conditions on a multivariate polynomial and applications. *Mathematics in Computer Science*, 1:177–207, 2007. 10.1007/s11786-007-0003-9. (cited on page 64)

[93] J.-K. Seong, E. Cohen, and G. Elber. Voronoi diagram computations for planar nurbs curves. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 67–77, New York, NY, USA, 2008. ACM. (cited on page 89)

[94] O. Setter, M. Sharir, and D. Halperin. Constructing two-dimensional voronoi diagrams via divide-and-conquer of envelopes in space. In M. L. Gavrilova and C. J. K. Tan, editors, *Transactions on computational science IX*, pages 1–27. Springer-Verlag, Berlin, Heidelberg, 2010. (cited on page 89)

[95] V. Sharma. Complexity of real root isolation using continued fractions. *Theor. Comput. Sci.*, 409(2):292–310, 2008. (cited on page 6)

[96] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Comput. Aided Geom. Design*, 10(5):379–405, 1993. (cited on page 7)

[97] M. Shub and S. Smale. Complexity of bezout's theorem i: Geometric aspects. *Journal of the American Mathematical Society*, 6(2):459–501, 1993. (cited on pages 21, 24, 32 and 33)

[98] F. Stenger. Computing the topological degree of a mapping in $\mathbf{R}^n$. *Numer. Math.*, 25(1):23–38, 1975. (cited on page 77)

[99] H. J. Stetter. Analysis of zero clusters in multivariate polynomial systems. In *Proceedings of the 1996 international symposium on Symbolic and algebraic computation*, ISSAC '96, pages 127–136, New York, NY, USA, 1996. ACM. (cited on pages 41 and 46)

[100] Z. Szafraniec. Topological degree and quadratic forms. *Journal of Pure and Applied Algebra*, 141(3):299 – 314, 1999. (cited on page 56)

[101] A. Tarski. *A decision method for elementary algebra and geometry*. Univ. of California Press, Berkeley, CA, 1951. (cited on page 64)

[102] P. Trébuchet. *Vers une résolution stable et rapide des équations algébriques*. PhD thesis, Université Pierre et Marie Curie, 2002. (cited on page 45)

[103] E. P. Tsigaridas. *Algebraic algorithms and applications to geometry*. PhD thesis, National Kapodistrian University of Athens, Aug 2006. (cited on page 6)

[104] E. P. Tsigaridas and I. Z. Emiris. On the complexity of real root isolation using Continued Fractions. *Theoretical Computer Science*, 392:158–173, 2008. (cited on pages 6, 25, 29 and 30)

[105] G. Tzoumas. *Computational geometry for curved objects. Voronoi diagrams in the plane*. PhD thesis, National & Kapodistrian Univ. Athens, 2009. In Greek. (not cited)

[106] J. van der Hoeven, G. Lecerf, B. Mourrain, P. Trebuchet, J. Berthomieu, D. N. Diatta, and A. Mantzaflaris. Mathemagix: The quest of modularity and efficiency for symbolic and certified numeric computation. *SIGSAM Communications in Computer Algebra*, 45(3), 2011. (cited on pages 34 and 78)

[107] A. van der Poorten. An introduction to continued fractions. In *Diophantine analysis*, pages 99–138. Cambridge University Press, 1986. (cited on page 25)

[108] J. von zur Gathen and J. Gerhard. Fast Algorithms for Taylor Shifts and Certain Difference Equations. In *Proc. Annual ACM ISSAC*, pages 40–47, 1997. (cited on page 14)

[109] M. N. Vrahatis. A short proof and a generalization of Miranda's existence theorem. *Proceedings of the American Mathematical Society*, 107(3):701–703, 1989. (cited on pages 22 and 23)

[110] X. Wu and L. Zhi. Determining singular solutions of polynomial systems via symbolic-numeric reduction to geometric involutive form. *J. Symb. Comput.*, 27:104–122, 2008. Accepted for publication. (cited on page 41)

[111] G. Xu, C. L. Bajaj, and C. I. Chu. Regular algebraic curve segments (ii)–interpolation and approximation. *Computer Aided Geometric Design*, 17(6):503 – 519, 2000. (cited on page 76)

[112] G. Xu, C. L. Bajaj, and W. Xue. Regular algebraic curve segments (i)–definitions and characteristics. *Computer Aided Geometric Design*, 17(6):485 – 501, 2000. (cited on page 76)

[113] J. Yakoubsohn. Approximating the zeros of analytic functions by the exclusion algorithm. *Numerical Algorithms*, 6(1):63–88, 1994. (cited on page 7)

[114] C. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000. (cited on page 25)

[115] Y. Yomdin and G. Comte. *Tame geometry with applications in smooth analysis*. LNM 1834. Springer-Verlag, 2004. (cited on page 21)

[116] Z. Zeng. The closedness subspace method for computing the multiplicity structure of a polynomial system. In D. Bates, G. Besana, S. Di Rocco, and C. Wampler, editors, *Interactions of Classical and Numerical Algebraic Geometry*, volume 496 of *Contemporaty Mathematics*, pages 347–362. Am. Math. Society, Providence, RI, 2009. (cited on pages 46 and 51)

Index