Geometric Algorithms for Algebraic Curves and Surfaces

Dissertation

zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

vorgelegt von

Michael Kerber

Saarbrücken
2009

Tag des Kolloquiums:
21. 12. 2009

Dekan der Naturwissenschaftlich-Technischen Fakultät I:
Prof. Dr. Joachim Weickert

Berichterstatter:
Prof. Dr. Kurt Mehlhorn, Max-Planck-Institut für Informatik, Saarbrücken
Prof. Laureano Gonzalez-Vega Ph.D, University of Cantabria, Santander, Spain
Prof. Chee Yap Ph.D., Courant Insitute, New York University, New York, USA

Mitglieder des Prüfungsausschusses:
Prof. Dr. Raimund Seidel (Vorsitzender)
Prof. Dr. Kurt Mehlhorn
Prof. Laureano Gonzalez-Vega Ph.D (per Videokonferenz)
Prof. Chee Yap Ph.D
Dr. Michael Sagraloff

Abstract

This work presents novel geometric algorithms dealing with algebraic curves and surfaces of arbitrary degree. These algorithms are exact and complete – they return the mathematically true result for all input instances. Efficiency is achieved by cutting back expensive symbolic computation and favoring combinatorial and adaptive numerical methods instead, without spoiling exactness in the overall result.

We present an algorithm for computing planar arrangements induced by real algebraic curves. We show its efficiency both in theory by a complexity analysis, as well as in practice by experimental comparison with related methods. For the latter, our solution has been implemented in the context of the CGAL library. The results show that it constitutes the best current exact implementation available for arrangements as well as for the related problem of computing the topology of one algebraic curve. The algorithm is also applied to related problems, such as arrangements of rotated curves, and arrangments embedded on a parameterized surface.

In $\mathbb{R}^3$, we propose a new method to compute an isotopic triangulation of an algebraic surface. This triangulation is based on a stratification of the surface, which reveals topological and geometric information. Our implementation is the first for this problem that makes consequent use of numerical methods, and still yields the exact topology of the surface.

The thesis is written in English.

Zusammenfassung

Diese Arbeit stellt neue Algorithmen für algebraische Kurven und Flächen von beliebigem Grad vor. Diese Algorithmen liefern für alle Eingaben das mathematisch korrekte Ergebnis. Wir erreichen Effizienz, indem wir aufwendige symbolische Berechnungen weitesgehend vermeiden, und stattdessen kombinatorische und adaptive numerische Methoden einsetzen, ohne die Exaktheit des Resultats zu zerstören.

Der Hauptbeitrag ist ein Algorithmus zur Berechnung von planaren Arrangements, die durch reelle algebraische Kurven induziert sind. Wir weisen die Effizienz des Verfahrens sowohl theoretisch durch eine Komplexitätsanalyse, als auch praktisch durch experimentelle Vergleiche nach. Dazu haben wir unser Verfahren im Rahmen der Softwarebibliothek CGAL implementiert. Die Resultate belegen, dass wir die zur Zeit beste verfügbare exakte Software bereitstellen. Der Algorithmus wird zur Arrangementberechnung rotierter Kurven, oder für Arrangements auf parametrisierten Oberflächen eingesetzt.

Im $\mathbb{R}^3$ geben wir ein neues Verfahren zur Berechnung einer isotopen Triangulierung einer algebraischen Oberfläche an. Diese Triangulierung basiert auf einer Stratifizierung der Oberfläche, die topologische und geometrische Informationen berechnet. Unsere Implementierung ist die erste für dieses Problem, welche numerische Methoden konsequent einsetzt, und dennoch die exakte Topologie der Oberfläche liefert.

Diese Dissertation ist in englischer Sprache verfasst.

Acknowledgments

I thank Kurt Mehlhorn for allowing me to be member of his very productive working group for the past three years. The level of freedom he admits his researchers encourages to choose reserach topics autnonomously – an invaluable experience for a young scientist.

I am especially indebted to Michael Sagraloff for his advice in all respects. Furthermore, I enjoyed a lot of fruitful discussions with Kurt Mehlhorn, Eric Berberich and Michael Hemmer. Pavel Emeliyanenko gave great assistance regarding the software part of this work. I also thank Arno Eigenwillig and Nicola Wolpert for introducing me to the field of geometric computing in which this work is embedded.

This work has been prepared using a LaTeX-template by Joachim Reichel, refined by Eric Berberich. Earlier drafts of the thesis have been read by Marc Bégin, Eric Berberich, Arno Eigenwillig, Pavel Emeliyanenko, Tobias Gärtner, Michael Hemmer, Kurt Mehlhorn, and Michael Sagraloff. I appreciate their numerous helpful suggestions. Finally, I thank Laureano Gonzalez-Vega and Chee Yap for accepting to review this thesis.

Schliesslich danke ich meinen Eltern für ihre Unterstützung meiner akademischen Laufbahn und Michaela für Ihre Liebe und Geduld.

Contents

1. Introduction **7**
1.1. Main contributions of this work . . . 9
1.2. Related work . . . 11
1.3. Outline . . . 13

2. Algebraic Foundations **15**
2.1. Topology . . . 15
2.2. Algebraic curves . . . 18
2.3. (Sub)resultants and Sturm-Habicht sequences . . . 26
2.4. Computation with polynomials . . . 38
2.5. Computation with algebraic numbers . . . 55
2.6. Computation with polynomials with bitstream coefficients . . . 74

3. Arrangements of Algebraic Plane Curves **93**
3.1. Arrangements by sweeping . . . 95
3.2. Algorithm for curve analysis . . . 107
3.3. Algorithm for curve pair analysis . . . 134

4. Implementation of Arrangements of Algebraic Plane Curves **147**
4.1. Implementation of curve and curve pair analysis . . . 147
4.2. Software design in CGAL . . . 156
4.3. Experiments . . . 165

5. Applications of Algebraic Arrangements **177**
5.1. A web application for visualizing algebraic arrangements . . . 177
5.2. Arrangements of rotated curves . . . 179
5.3. Arrangements on tori and Dupin cyclides . . . 187
5.4. Further applications . . . 196

6. Stratification and Triangulation of Algebraic Surfaces **199**
6.1. The n-k-arrangement . . . 202
6.2. Z-fibers and cell decomposition . . . 206
6.3. Cell adjacencies . . . 208
6.4. Triangulation . . . 213
6.5. Implementation and experiments . . . 220
6.6. Bounds for the size of isocomplexes . . . 223

Bibliography **231**

Index **243**

A. Realization of Trigonometric Functions **247**
A.1. Approximating π 247
A.2. Approximating sin 248
A.3. Approximating arcsin 249

B. Curriculum Vitae **251**

C. List of Publications **253**

Nobody untrained in geometry may enter my house!

Inscription over the door of Plato's academy

1 Introduction

Computational geometry is a sub-discipline of algorithmics, dealing with problems that can be expressed in geometric terms. Its popularity is mainly based on the wide applicability of its results in several areas, for instance in computer aided design, computer graphics, robotics, geographic information systems, and biology. Mostly, the input of geometric algorithms consists of linear objects like points, lines, (hyper)planes, or polygons. The advantages are clear: linear objects can be processed faster than non-linear ones and the simpler structure eases the design of algorithms and their analysis. However, real problems are often of a non-linear nature, and can only be modeled approximately by linear objects. Designing algorithms that accept circles or circular arcs as input increases the accuracy, but also might not suffice in all cases.

This work presents algorithms for geometric objects in two and three dimensions that are defined by arbitrary algebraic equations. An algebraic curve in $\mathbb{R}^2$ is the set of solutions of an equation $f = 0$, where f is some (integer or rational) polynomial in two variables. This class of curves covers lines (defined by a linear equation) as well as circles (defined by an equation of degree 2) as its simplest representatives. Analogously, an algebraic surface is defined by $F = 0$, where F is a polynomial in three variables. The two main problems addressed in this thesis are:

1. Given a set of planar algebraic curves $C_1, \ldots, C_m$, compute a geometric-topological description of the arrangement induced by these curves. More precisely, compute a doubly connected edge list (DCEL) that captures how the plane is decomposed by $C_1, \ldots, C_m$ into 0-, 1-, and 2-dimensional components, called *vertices*, *edges*, and *faces*. See also Figure 1.1 (left).
2. Given an algebraic surface S, compute a geometric-topological description in terms of a triangulation T. In particular, T and S should be isotopic (i.e., S can be transformed continuously into T without topological changes). See also Figure 1.1 (right).

It is important to note that both algorithms realize non-continuous functions: small changes in the input can lead to profound changes in the resulting combinatorial structure (i.e., the DCELor the triangular mesh). This is a characteristic property in geometric computation and raises the question of robustness.

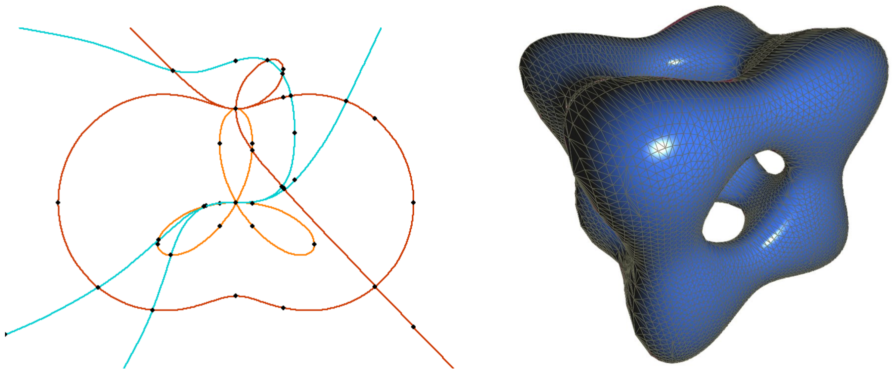


Figure 1.1. On the left: An arrangement of three algebraic plane curves. On the right: A triangulation of an algebraic surface called "tangle-cube". Both pictures are produced based on the algorithms presented in this thesis.

Geometric algorithms are often composed of a set of basic geometric predicates and constructions, also called *primitives*. Two typical examples of such primitives are: *Given three points p, q, and r, determine whether the triangle pqr has clockwise or counterclockwise orientation, or is degenerated* (if the points lie on a common line). *Given two lines, construct their intersection point.* Correctness of an algorithm is usually proved under the assumption that all primitives deliver the correct result. Moreover, many algorithms assume a *generic* input. The precise meaning of genericity depends on the algorithm, but a very common condition is that no three input points lie on the same line. The reason for this assumption is mainly convenience since handling special cases makes the description lengthy and complicates theoretical considerations.

When it comes to a concrete implementation of geometric algorithms, the question of how to realize the geometric primitives arises. Note that already fairly simple predicates are problematic: intersecting a line and a circle in general yields a point with irrational coordinates, and it is non-trivial to represent and process such a point in subsequent predicates. Evaluating all primitives using fixed-precision floating-point numbers seems appealing as a way of circumventing this problem and doubtlessly works for many instances – but not for all (Figure 1.2). The induced rounding errors can also have disastrous effects due to the non-continuous nature of geometric algorithms (see above). It is not only that the result might differ substantially from the exact solution – if conditionals inside the algorithms depend on results of primitives, the wrong computation branch might be executed. The effects are infinite loops or program crashes. Although such effects can be prevented by modifying the algorithm, known methods only apply to specific problems and do not extend to a general framework.

We will follow another approach: the *Exact Geometric Computation paradigm* (EGC paradigm) . It simply states that the implementation of a geometric primitive is required to deliver the mathematically correct result in all cases. It follows that a geometric algorithm based on these primitives returns the mathematically correct result, which is clearly a pleasant property for the user of such an algorithm. The drawback is the performance penalty caused by the paradigm; even for linear objects, data types modeling arbitrary rational numbers become necessary for the computation. Even worse, when dealing with

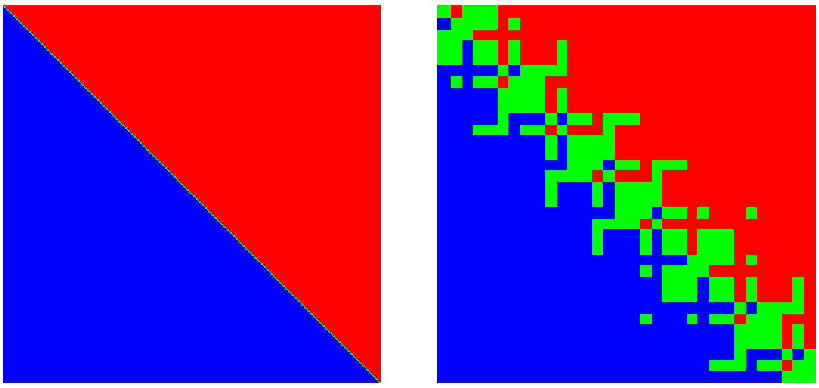


Figure 1.2. Example for the instability of floating-point computations, taken from [KMP+08]. Let $q := (12, 12), r := (24, 24)$, and $p := (0.5 + 2^{-64}x, 0.5 + 2^{-64}y)$. The picture shows a 256×256 grid, where the pixel (x, y) is drawn red if the points p, q, r are in clockwise orientation, blue if they are in counterclockwise orientation, and green if they are colinear (this can be expressed as the sign of a determinant in the coordinates of p, q, and r). On the left, we see the exact result, on the right, we see the same when the geometric predicate is evaluated using `long double` in C++. Note that not just the three points are wrongly assumed to be colinear (red and blue points become green for floating-point arithmetic), but also the orientation changes completely (red points become blue, and vice versa).

algebraic objects, one needs to represent and process algebraic numbers and polynomials in an exact way. In particular, one has to compute the roots of uni- and multivariate polynomial equations, which leads to the field of symbolic computation and computer algebra.

1.1. Main contributions of this work

EGC solutions for geometric primitives in the plane are presented for the case of points and segments defined by algebraic equations. Although the name "primitives" suggests that the operations are rather simple, this is not at all the case. We reduce their realization to the geometric-topological analysis of single curves and of pairs of curves (for the arrangement problem), termed *curve analysis* [EKW07] [Ker06][1] and *curve pair analysis* [EK08a], respectively. It is worth emphasizing that our solution works for curves of arbitrary degree and covers all possible special cases. These analyses can be considered as special cases of a *cylindrical algebraic decomposition* (*cad*) (see related work below). Readers unfamiliar with *cad*s might imagine the curve analysis as an algorithm for *topology computation*, which means computing an embedded straight-line graph that is isotopic to the curve (with some additional geometric properties), and the curve pair analysis as a similar method for the union of two curves.

The algorithmic goal for both types of analyses (as well as for the *surface stratification* following below) is to keep the amount of costly symbolic computations as low as

[1] This algorithm was the subject of the master's thesis of the author.

possible. The EGC paradigm does not restrict our choice of methods *inside* a geometric primitive – we can use any technique that works well, as long as it is guaranteed to produce the exact overall result. We prefer to apply combinatorial and (validated) floating-point methods because they are faster than symbolic computations. Some substeps can always be performed approximately. If the analyzed object is "good-natured" (e.g., no singular points, in generic position), even more approximate methods are applied successfully. This is done adaptively, meaning the algorithm decides internally whether to apply symbolic or non-symbolic methods. The effect is that the analysis works fast for simple examples and spends more time only in the presence of complicated features.

The certainly most significant tool for practical efficiency is the *bitstream-Descartes method* [Eig08] [MS09]. It computes the roots of a square-free polynomial with algebraic coefficients only by approximating the coefficients. We present two generalizations of the method that allow its application to non-square-free polynomials as well. These variations have already been discussed in the author's master's thesis, but we give a more detailed description in this thesis, including a complexity analysis, based on work by Mehlhorn and Sagraloff [MS09].

We examine the asymptotical complexity of both the curve and curve pair analysis in terms of bit complexity. The obtained bound is the same in both cases and matches the currently best-known bound for topology computation of algebraic curves by Diochnos et al. [DET07]. Their bound is achieved by a purely symbolic method; we obtain the same result for an algorithm that is mainly based on isolating and refining real roots of polynomial systems, which is widely acknowledged to be faster in practice. Also, the curve analysis computes additional geometric information about the analyzed curve, which is not the case for most other approaches that only compute the topology. Crucial for our analysis is a novel result for the subproblem of approximating an algebraic number to a given precision [Ker09b].

Our algorithms are implemented in C++ in the context of the CGAL library. We provide experimental results of our approach and compare it with related algorithms for computing curve topology and *cad*s. The result is an overall satisfying performance, in some situations much better than other approaches, especially for curves with large coefficients. Also, the adaptive behavior of our fast approximate methods can be observed in the experimental results. Furthermore, our approach outperforms existing arrangement algorithms for restricted classes of algebraic curves, such as conics or rational functions, which are currently available in the CGAL library. We conclude that our design choice of cutting back symbolic computations is successful and that our implementation is in a mature state. An integration into a public release of CGAL is planned; in the meantime, we run a publicly available web-server that allows the computation and exploration of arrangements of algebraic curves interactively [EK08c].

We also implemented some variants of our arrangement algorithm. It is possible to rotate algebraic curves to certain angles and compute the (exact) arrangement of those rotated curves. Another generalization computes arrangements not in the plane, but on a (parameterizable) surface embedded in $\mathbb{R}^3$, in particular on tori and Dupin cyclides [BK08] [BFH$^+$09a]. Both approaches profit from the generic design of our software, technically achieved by template code in C++, which allows combining it with other CGAL packages. Currently ongoing work uses our software also for other problems than computing arrangements, such as computing Boolean set operations, or Voronoi diagrams of lines in $\mathbb{R}^3$.

Last but not least, this thesis presents an algorithm to compute an isotopic triangulation for an arbitrary algebraic surface [BKS09]. This is the first exact solution to this problem that makes consequent use of (certified) numerical methods. It is based on a stratification that decomposes the surface into vertices, edges, and patches [BKS08]. The result of this decomposition is similar to a *cad* of the surface but with fewer cells. Again, an implementation of the method has been done and shows the success of our algorithmic design. The resulting triangulation has the same complexity as a *cad*; an interesting question is whether a less complex isotopic triangulation exists. We derive lower bounds for the output graph in the topology computation for curves as well as for the triangulation problem.

1.2. Related work

The foundations of computational geometry are covered by textbooks at the undergraduate level [dBvKOS00] [PS85] [Meh84]; comprehensive collections have appeared in [GO97] [SU00].

Robustness problems using floating-point numbers instead of exact computation have been known for a long time [For70], also in a geometric context; see [Hof89] for an early survey. Kettner et al. [KMP+08] present several examples of what can go wrong and why. Schirra [Sch00] and Yap [Yap97a] survey various approaches to cope with the pitfalls of imprecise computations in geometry. For instance, one possibility is to maintain certain topological properties during the algorithm, even if numerical computations would suggest otherwise [Sug99], but this requires a re-design of each considered algorithm. Another approach is to identify a set of *axioms* that a primitive has to satisfy in order to make the overall algorithm correct and to choose a suitable precision for the execution [Sch93]. This requires a separate analysis of each considered algorithm. A more recent technique is *controlled perturbation*, where the input is perturbed slightly, such that degenerate situations during the computation are prevented, and the exact result (for the perturbed input) can be achieved by fixed floating-point computations. See [HS98] [Raa99] [FKMS05] [HL03] for applications to specific problems; a general scheme handling a larger class of problems has been presented in [MOS06]. Although a nearby solution is certainly sufficient in a lot of applications, controlled perturbation techniques are inadequate if an exact solution is required.

Our work concentrates on the exact geometric computation paradigm, made popular by Yap [Yap97b]. Data types for modeling integers and rational numbers of arbitrary size are indispensable for this approach; they are provided by several libraries, such as GMP,[2] LEDA [MN00],[3], and CORE [KLPY99]. Several techniques to evaluate primitives in an exact *and* efficient way have been studied. The result of a geometric predicate often boils down to the sign of a polynomial expression evaluated at the coordinates of the involved points. *Floating-point filters* evaluate the expression with some fixed precision (usually hardware precision) and if the result has a greater absolute value than the maximal error caused by the computation, the sign is determined [MN00, §9.7] [FV96] [BEPP97]. For the special case of integer coordinates, these methods can also serve as a zero test if the maximal error is smaller than 1. If the floating-point filter fails, one either uses exact arithmetic to get the result or increases the precision and retries the approximate computation. This

[2] `http://gmplib.org/`
[3] `http://www.algorithmic-solutions.com/`

is called *adaptive precision arithmetic* [She96]. Closely related are the adaptive precision methods based on *interval arithmetic*. We will introduce this concept in detail in Section 2.5.4 and discuss references therein. All these techniques apply to predicates but not to constructions; Funke and Mehlhorn [FM02] introduce a lazy-evaluation technique that allows one to filter geometric constructions as well. The idea is to avoid the computation of the explicit coordinates during the construction and to compute it afterwards only if needed.

Non-linear computational geometry, especially with algebraic objects, has become an active field of research during the past years. An IMA workshop in 2007 was dedicated to this subject [EST09], bringing together the areas of computational geometry and (real) algebraic geometry. Also, two EU-funded projects covered this topic; the first one, called *Effective Computational Geometry for Curves and Surfaces* (ECG)[4] ran from 2001–2004; see [BT06] for a collection of results. The successor project *Algorithms for Complex Shapes with Certified Topology and Numerics*[5] (2004–2007) extended the success of the results, as the name states, with a focus on certified computation. Our work is embedded in the context of this project. Some of our results have been published in a preliminary form as technical reports of the ACS project.

A big advantage of the EGC paradigm is its general approach that applies to any geometric algorithm. Its success is documented by the popularity of the software libraries LEDA[6] (mostly for linear objects) and CGAL [CGA08]. The integration of non-linear objects into CGAL was substantially brought forward by the two EU projects mentioned above. During this period, the library EXACUS [BEH+05] for exact algorithms for curves and surfaces was established at MPI,[7] and its contents have been (and are still being) merged into CGAL.

Computer algebra, or *symbolic computation*, is a discipline on the edge between mathematics and computer science that deals with algorithms for algebraically defined objects, in particular, uni- and multivariate polynomials and algebraic numbers. An early collection of results in that area is [BCL82]; textbooks covering this area have been written by Basu et al. [BPR06], Geddes et al. [GCL92], von zur Gathen and Gerhard [vzGG97], and Yap [Yap00]. We will frequently cite these books in the algebraic parts of this thesis.

Symbolic methods can be avoided completely in geometric computations, in principle, with *constructive separation bounds*: for an algebraic expression, say E, and an argument p, one computes a value L such that either $|E(p)| \geq L$, or $E(p) = 0$. Then, $E(p)$ is evaluated approximately using floating-point arithmetic until either its absolute value, plus some error term, is smaller than L (and thus $E(p) = 0$), or it is guaranteed to be either positive or negative. This idea seems to go back to Mignotte [Mig82]. Li and Yap [LY01] and Burnikel et al. [BFM+01] give methods to construct such bounds, and the libraries CORE (CORE::Expr) and LEDA (leda::real) provide data types to use such separation bounds in practice. However, the derived lower bounds have to assume the worst case and tend to be too pessimistic. Indeed, we observed that the use of these data types renders our implementation impracticable even for medium-sized instances. Thus, our goal should be stated more carefully: we want to reduce the number of symbolic computations in our algorithms, but without applying constructive separation bounds.

[4] `http://www-sop.inria.fr/prisme/ECG/`
[5] `http://acs.cs.rug.nl/index.php`
[6] LEDA is not restricted to geometric algorithms but has a much larger scope; see [MN00]
[7] http://www.mpi-inf.mpg.de/projects/EXACUS/

The *cylindrical algebraic decomposition* (*cad*) [Col75] [CJ98] is a data structure that originates in quantifier elimination problems. For a set S of polynomials in d variables, it constitutes a decomposition of $\mathbb{R}^d$, such that each cell has invariant sign for each polynomial in S. Moreover, the projection of two cells to $\mathbb{R}^{d-1}$ by eliminating one variable either yields the same region or two disjoint regions (hence the name "cylindrical"). Some (geometric) applications of *cad*s are mentioned in [Mis97]. Arnon et al. [ACM84a] presented an algorithm to compute a *cad* in $\mathbb{R}^d$. Their approach is purely symbolic, and numerical methods have been proposed to speed up the algorithm [Str06] [CJK02] [Bro02]. These optimizations work in a way similar to that of floating-point filters, meaning they try to get the result using some fixed precision and fall back to the purely symbolic method in case of failure. Together with a method to compute the adjacencies between cells of the *cad* in $\mathbb{R}^2$ and $\mathbb{R}^3$ [ACM84b] [ACM88] [MC02], any such *cad* algorithm can be modified to yield an EGC-implementation for curve and curve pair analysis and for the triangulation of surfaces. Vice versa, our methods can also be seen as alternative variants for computing *cad*s, for the special cases of one or two curves or of one surface. Our methods profit from fast numerical computations for any instance and never have to fall back to a completely symbolic method – a main difference to all previous approaches. We will point out more similarities and differences to *cad* approaches in the corresponding chapters.

Related work on arrangements is discussed in Chapter 3, and on stratifications and triangulations in Chapter 6.

1.3. Outline

Chapter 2 continues with the mathematical foundations needed in this thesis: We introduce algebraic curves and their most important geometric properties for our needs. Also, we introduce the toolbox of algorithms dealing with polynomials and algebraic numbers, which is used frequently in our algorithms. This part is relatively extensive since we establish complexity bounds for all presented algorithms.

Chapters 3 and 4 describe our approach computing arrangements of algebraic curves. We decided to split the treatment into an "algorithmic" and an "implementation" part: Chapter 3 defines the set of primitives needed for arrangements, how they relate to curve analysis and curve pair analysis, and contains a complete solution for both, including a complexity analysis. However, several practical optimizations are not covered by the treatment, in order not to make the description (and the analysis) too detailed. Chapter 4 explains these optimizations, describes the software design of our implementation, and presents experimental results.

Some applications of the arrangement algorithm are discussed in Chapter 5: a webserver to compute and explore arrangements interactively, a method to compute arrangements of rotated curves, and an algorithm to compute arrangements on parameterizable surfaces, exemplified with Dupin cyclides.

Chapter 6 discusses an approach for computing a stratification of an algebraic surface and refines the derived decomposition into a triangulation of the surface. Some experimental results are presented and lower and upper bounds on the output size of topology computation are given for algebraic curves and surfaces.

If in other sciences we should arrive at certainty without doubt and truth without error, it behooves us to place the foundations of knowledge in mathematics.

Roger Bacon

2 Algebraic Foundations

In this chapter, we will establish the tools necessary to formulate the geometric algorithms for algebraic curves and surface in the subsequent chapters. Not surprisingly, the number of prerequisites is quite big, since algebraic objects are a very general class of geometric objects, and non-trivial to work with. We mainly concentrate on algebraic curves in this chapter, but many results extend directly to algebraic surfaces as well.

Before we start with algebraic objects, we introduce some notation from topology (Section 2.1) that will be convenient in this thesis. Then, we introduce algebraic curves, and summarize their main properties in Section 2.2. Subresultants and Sturm-Habicht sequences, our main symbolic tools to investigate the geometry of algebraic curves, are the subject of Section 2.3.

Afterwards, we turn to computational issues. First, we introduce algorithms for integer polynomials in Section 2.4, for instance, basic arithmetic, computation of greatest common divisors, root isolation, etc. In Section 2.5, we explain how to represent and work with arbitrary algebraic numbers, for instance, how to compare them, or how to approximate them to any precision. Finally, in Section 2.6, we rediscuss the root isolation (and the root refinement) problem in the situation where a polynomial's coefficients are given only approximately.

For all algorithms in Sections 2.4–2.6, we give worst-case bounds for the number of performed bit operations.

2.1. Topology

We will now establish some well-known concepts from topology. Although the material can be covered in a more abstract way, we will only consider the case in which the underlying space is $\mathbb{R}^d$ (and in fact, we only care about $d = 2$ and $d = 3$), equipped with the usual Euclidean distance $\|p - q\|_2$ for two points $p, q \in \mathbb{R}^d$.

Definition 2.1.1 (open set). A set $S \subset \mathbb{R}^d$ is *open* if for any $s \in S$, there exists a ball $\{t \in \mathbb{R}^d \mid \|t - s\|_2 < \varepsilon\}$ of radius $\varepsilon > 0$ that is completely contained in S.

For a subset $X \subset \mathbb{R}^d$, a set $S \subset X$ is *open with respect to* X, if there exists an open set $S_0 \subset \mathbb{R}^d$ such that $S = S_0 \cap X$.

For instance, $[0, \frac{1}{2})$ is an open set with respect to the unit interval $[0, 1]$.

Definition 2.1.2 (continuous mapping). Let X, Y be two subsets of $\mathbb{R}^d$. A map $f : X \to Y$ is called *continuous*, if for any set $M_1 \subset Y$ that is open with respect to Y, there is a set $M_2 \subset X$ open with respect to X such that $f^{-1}(M_1) = M_2$.

An intuitive description is that for a continuous mapping, the preimage of an open set is again open. But one should be careful about this description, since "open" relates to the corresponding domain.

Definition 2.1.3 (homeomorphism). A map $f : X \to Y$ is called a *homeomorphism*, if it is bijective, continuous, and the inverse map f^{-1} is also continuous. In this case the sets X and Y are called *homeomorphic*.

The idea behind homeomorphic sets is that points are "separated" in X if and only if they are "separated" in Y. We give the standard example of a non-homeomorphic map: Let S^1 be the unit circle and consider

$$f : [0, 2\pi) \to S^1; x \mapsto (\cos x, \sin x).$$

This map is bijective and continuous, but the inverse is not continuous. For instance, consider the open set $M_1 := [0, \pi)$ with respect to $[0, 2\pi)$. The preimage of the inverse function on M_1 is nothing but $f(M_1)$. This is the upper part of the circle, excluding the point $(-1, 0)$, and including $(1, 0)$. Obviously, this is not an open set with respect to S^1. In fact, there is no homeomorphism between the two sets.

Definition 2.1.4 (connected set). Let $X \subset \mathbb{R}^d$. X is *connected* if it cannot be decomposed into two (or more) open sets with respect to X. A maximal connected set $M \subset X$ is called a *connected component*.

A homeomorphism $f : X \to Y$ maps connected components of X to connected components of Y. However, the relations between connected components as embeddings into $\mathbb{R}^d$ might change: The sets $S^1 \cup \{(0, 0)\}$ and $S^1 \cup \{(0, 2)\}$ are homeomorphic, but there is a notable difference between them: In the former case the isolated point lies inside the circle and in the latter it lies outside the circle. We consider this as a change in the topology of the point set since, intuitively, we cannot transform the one set into the other without moving the point over the circle. We next derive a formal concept that is stronger than homeomorphism and formalizes this intuition.

Definition 2.1.5 (isotopic). Let $X, Y \subset \mathbb{R}^d$. We say that X and Y are *isotopic* if there exists a continuous map $H : [0, 1] \times X \to \mathbb{R}^d$ with the following properties:

1. For any $t \in [0, 1]$, $H_t(x) := H(t, x)$ is a homeomorphism between X and $I_t := H_t(X)$, the image of H_t.
2. $H_0(\,\cdot\,) = \mathrm{id}_X(\,\cdot\,)$
3. $I_1 := Y$, which means that H_1 is a homeomorphism of X and Y.

In this case, H is called an *isotopy* between X and Y.

Note this additional requirement, compared to homeomorphic sets X and Y: Isotopic sets must be “connected” by homeomorphism, which prevents the relative position of connected components changing. As an example, consider again $S^1 \cup \{(0,0)\}$ and $S^1 \cup \{(0,2)\}$. They are not isotopic, because the isolated point either crosses the circle for some $t \in [0,1]$ (which implies that H_t is not homeomorphism) or it “jumps” for some $t \in [0,1]$ (which implies that H is not continuous).

We now provide some simple examples of isotopic sets.

- Translations: Let $v \in \mathbb{R}^d$, and $f : \mathbb{R}^d \to \mathbb{R}^d, x \mapsto x + v$. The sets $X \subset \mathbb{R}^d$ and $f(X)$ are isotopic. An isotopy is given by $H(t,x) = x + tv$.
- Rotations: Let $\varphi \in [0, 2\pi)$, and

$$f : \mathbb{R}^2 \to \mathbb{R}^2, \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

The sets $X \subset \mathbb{R}^2$ and $f(X)$ are isotopic. An isotopy is given by

$$H \begin{pmatrix} t \\ x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \cos t\varphi & \sin t\varphi \\ -\sin t\varphi & \cos t\varphi \end{pmatrix}.$$

- Shear transformations: Let $s \in \mathbb{R}$, and

$$f : \mathbb{R}^2 \to \mathbb{R}^2, \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x + sy \\ y \end{pmatrix}.$$

The sets $X \subset \mathbb{R}^2$ and $f(X)$ are isotopic. An isotopy is given by

$$H \begin{pmatrix} t \\ x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x + tsy \\ y \end{pmatrix}.$$

We will also need the notion of a simplicial complex.

Definition 2.1.6 (simplex). The $k+1$ points $p_0, \ldots, p_k \in \mathbb{R}^d$ are *affinely independent* if the k vectors $p_i - p_0$, $i = 1, \ldots, k$ are linearly independent in $\mathbb{R}^d$. It follows necessarily that $k \leq d$. A *simplex* σ of dimension k is the convex hull of $k+1$ affinely independent points $p_0, \ldots, p_k$. We also say that $p_0, \ldots, p_k$ span σ. A *subsimplex* of σ in $\mathbb{R}^d$ is a simplex spanned by a non-empty subset of $\{p_0, \ldots, p_k\}$.

Definition 2.1.7 (simplicial complex). A *simplical complex* $\mathcal{C}$ is a set of simplices $\sigma_1, \ldots, \sigma_s$ such that each subcomplex τ of any σ_i is also in S, and such that for any pair of simplices, $\sigma_i \cap \sigma_j$ is either empty or is a simplex in S. The dimension of a simplicial complex is the maximal dimension of the contained simplices. The 0-dimensional simplices are called *vertices* of the complex.

Often, abstract simplicial complexes are considered; they are not embedded into Euclidean space. However, we will only consider embedded complexes and we will even identify a complex and its induced point set, that is, $S = \bigcup_{i=1}^{s} \sigma_i$.

Definition 2.1.8 ((stable) isocomplex). For a point set $X \subset \mathbb{R}^d$, an *isocomplex* $\mathcal{C}$ is a simplicial complex that is isotopic to X. It is called *stable* if there exists an isotopy

$H : [0,1] \times \mathcal{C} \to X$ such that $H(t,v) = v$ for any $t \in [0,1]$ and for any vertex v of $\mathcal{C}$. In particular, vertices of $\mathcal{C}$ belong to X. If we do not force the stability condition, we talk about a *general isocomplex*

In $\mathbb{R}^3$, an isocomplex is usually called a *triangulation* of the surface. This thesis contains algorithms to compute stable isocomplexes for algebraic curves and surfaces.

2.2. Algebraic curves

We assume that the reader is familiar with basic algebraic concepts like domains, fields, and polynomial rings. We only expose the theory of algebraic curves as much as we need it for our results, and especially focus on (real) geometric aspects; for a much deeper introduction to the material, consult the following textbooks [Wal50] [Gib98] [BK81]. Some of the facts can be stated in a more general way (e.g., for arbitrary dimension), but we have tried to keep them as close as possible to the concrete problems at hand. A large part of the material has already appeared in [Ker06, §§2.1 and 2.2].

2.2.1. Basic definitions and notation

This section reminds the reader of basic definitions and results from algebra that are used in this work. This material can be easily found in introductory textbooks on algebra (as [Lan93] [Bos04] [Wol96] [vdW71]).

A *domain* D is a commutative ring with unity where $ab = 0$ implies that a or b is zero. An element $a \in D$ *divides* $b \in D$ if there is an element $c \in D$ such that $ac = b$. We also use the notations that a is a *divisor* of b or that b is *divisible* by a. A *unit* in D is an element that divides each element of the ring, or equivalently, an element that has a multiplicative inverse. We call $a \in D$ and $b \in D$ *associates* if $a = ub$ for a unit $u \in D$. An element $a \in D$ is *irreducible* if $a = bc$ implies that b or c is a unit.

A domain D is called *factorial* if each $a \in D$ can be, up to associates, factorized uniquely into irreducible elements. For instance, $\mathbb{Z}$ is factorial and its irreducible elements are the prime numbers. A factorial domain is also called a *unique factorization domain*, or *UFD*, for short. In a UFD, the *greatest common divisor* $\gcd(a_1, \ldots, a_n)$ (with $n \geq 2$) is well-defined and unique up to associates.[8]

For a domain D, the polynomial ring $D[t]$ in the indeterminate t consists of polynomials f with $f = 0$ or $f = \sum_{i=0}^{n} a_i t^i$ with $a_i \in D$ and $a_n \neq 0$. In the latter case, n is the *degree* of f, or in short, $\deg f = n$. We set $\deg(0) = -\infty$. For $f \neq 0$, $a_0, \ldots, a_n$ are called the *coefficients* of f, and we write $\mathrm{coef}_i(f) := a_i$ for $i = 0, \ldots, n$, and $\mathrm{coef}_i(f) = 0$ for $i > n$. a_n is the *leading coefficient* of f, $a_n = \mathrm{lcf}(f)$. The *content* of $f \in D[t]$ is the gcd of its coefficients, $\mathrm{cont}(f) := \gcd(a_0, \ldots, a_n)$. A polynomial is called *primitive* if its content is 1. The *primitive part* of a polynomial f is the polynomial $\mathrm{pp}(f) := \frac{f}{\mathrm{cont} f} \in D[t]$. f is called *square-free* if there exists no $g \in D[t]$ with $\deg g \geq 1$ such that g^2 divides f. A *square-free part* $\overline{f}$ of f is a square-free polynomial dividing f with positive leading coefficient such that any square-free polynomial that divides f also divides $\overline{f}$.[9]

[8] This "up to associates" restriction can be relaxed easily by choosing a representative for each equivalence class of associates. For instance, if $D = \mathbb{Z}$, always choose the positive element, and if $D = \mathbb{Q}[t]$, choose the element with leading coefficient one.

[9] As for the gcd, the square-free part can be made unique by choosing a suitable representative system.

A famous lemma by Gauss states that if D is a UFD, then $D[t]$ is also a UFD. Consequently, the gcd is also well-defined for polynomials over a UFD. The *derivative* f' of f is the polynomial $f' = \sum_{i=0}^{n-1}(i+1)a_{i+1}t^i$. One can show that $f \in D[t]$ is square-free if and only if $\gcd(f, f')$ is a constant, which means of degree 0. Moreover, the square-free part of f is given by $\overline{f} = \frac{f}{\gcd(f,f')}$, up to a constant factor.

The *field of fractions* Q of a domain D is the smallest field containing D.[10] Elements of Q can be expressed by fractions $\frac{a}{b}$ with $a, b \in D$ and $b \neq 0$. It is well known that a polynomial in $D[t]$ is irreducible as an element in $Q[t]$ if and only if it is irreducible in $D[t]$. Likewise, $f \in D[t]$ is square-free if and only if it is square-free while considered as an element of $Q[t]$.

Let $K \supset D$ be a field that contains D. A polynomial $f \in D[t]$ induces a map $K \to K$ by mapping $x_0 \in K$ to $\sum_{i=0}^{n} a_i x_0^i \in K$. We denote the function value by $f(x_0)$. A *root* α of f is an element such that $f(\alpha) = 0$. The *algebraic closure* $\overline{Q} \supset Q$ is the smallest field containing Q such that any polynomial $f \in Q[t]$ (and thus any polynomial $f \in D[t]$) has a root in $\overline{Q}$. By the fundamental theorem of algebra, each polynomial $f \in D[t]$ can be written as

$$f = \operatorname{lcf}(f)\prod_{i=1}^{m}(t - \alpha_i)^{e_i}$$

with $\alpha_i \in \overline{Q}$ and $e_i \geq 1$ such that $\sum_{i=0}^{m} e_i = n$. The integer e_i is called the *multiplicity* of the root α_i, $\operatorname{mult}(\alpha_i, f) = e_i$, and $\operatorname{mult}(x, f) = 0$ if x is not a root of f. A root α is called *simple* if its multiplicity is one, and *multiple* otherwise. For $f, g \in D[t]$, their gcd over Q equals their gcd over $\overline{Q}$. Thus, f is square-free if and only if all its roots are simple. For $K \supset D$, we call an element $\alpha \in K$ *algebraic* over D, if it is a root of a polynomial $f \in D[t]$. The algebraic element over $\mathbb{Z}$ are called the *algebraic numbers*.

A polynomial ring $D[t_1, \ldots, t_d]$ with indeterminates $t_1, \ldots, t_d (d \geq 1)$ consists of elements of the form

$$\sum_{\substack{j_1, \ldots, j_n \geq 0 \\ j_1 + \ldots + j_d \leq n}} a_{(j_1,\ldots,j_d)} t_1^{j_1} \cdots t_d^{j_d}$$

with $a_{(j_1,\ldots,j_d)} \in D$ where at least one $a_{(j_1,\ldots,j_d)} \neq 0$ with $j_1 + \ldots + j_d = n$. The $a_{(j_1,\ldots,j_d)}$ are the *scalar coefficients*, and the expressions $t_1^{j_1} \cdots t_d^{j_d}$ are the *monomials* of f. The *total degree* $\deg f := \deg_{tot} f$ of f is n. Obviously, these definitions generalize the definitions for $D[t]$ from above. We call polynomials with $d = 1, 2, 3$ *univariate*, *bivariate* and *trivariate*, respectively. We also use the term *multivariate* for $d > 1$. Note that for a UFD D, multivariate polynomial rings over D are also UFDs, again by Gauss' lemma.

A multivariate polynomial $f \in D[t_1, \ldots, t_d]$ has different interpretations. On the one hand, it is a polynomial built of scalar coefficients in D and monomials, as just described. On the other hand, it can be considered as a univariate polynomial in any t_i, with coefficients in the domain $D[t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_d]$. We denote by $\deg_{t_i}$ the degree of this univariate polynomial in t_i; clearly $\deg_{t_i} f \leq \deg_{tot} f$. Also, we set $\frac{\partial f}{\partial t_i}$ for the derivative f' as polynomial in t_i. In this thesis, we will frequently use both interpretations; to avoid misunderstandings, we will often use expressions like "the multivariate polynomial f" or $f \in D[t_1, \ldots, t_d]$ for the multivariate interpretation, and "the polynomial f, considered as a polynomial in t_i", or $f \in D[t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_d][t_i]$ for the univariate interpretation.

[10] More precisely: Any field that embeds D also embeds Q.

2.2.2. Definition and properties of algebraic curves

Definition 2.2.1 (vanishing set). Let $f \in \mathbb{K}[x_1, \ldots, x_n]$ be a polynomial, and $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. The *vanishing set* (also called *zero set*, or *zero locus*) of f in $\mathbb{K}$ is defined as

$$V_{\mathbb{K}}(f) = \{(a_1, \ldots, a_n) \in \mathbb{K}^n \mid f(a_1, \ldots, a_n) = 0\}.$$

For convenience, we set $V(f) := V_{\mathbb{R}}(f)$ and call it the vanishing set of f.

It is immediately apparent that

$$V(fg) = V(f) \cup V(g) \tag{2.1}$$
$$V(cf) = V(f)\ \forall c \in \mathbb{R} \setminus \{0\}. \tag{2.2}$$

Definition 2.2.2 (algebraic curve). A *real planar algebraic curve*, or just *algebraic curve*, is the vanishing set of a non-zero bivariate polynomial $f \in \mathbb{Z}[x, y]$. We also call $V(f)$ the curve *induced* by f, and say that f is the *defining polynomial* for $V(f)$. We also say that an algebraic curve $V(f)$ is of degree n if $\deg_{tot} f = n$. For $\alpha \in \mathbb{R}$, we call $V(f(\alpha, y))$ the *fiber* of f (or $V(f)$) at α.

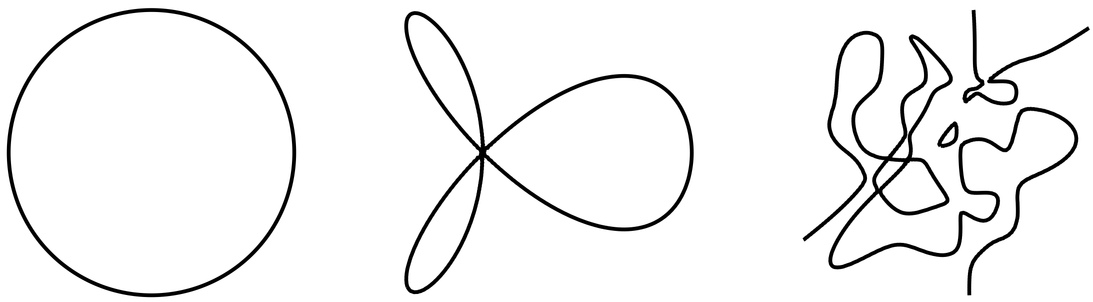


Figure 2.1. The curves $V(x^2 + y^2 - 1)$, $V(2x^4 + y^4 - x^3 + xy^2)$, and a randomly generated curve of degree 10.

Some examples for algebraic curves can be found in Figure 2.1. Note that different polynomials might induce the same curve. In particular:

Proposition 2.2.3. *Let $f \in \mathbb{Z}[x][y]$ and let $\overline{f}$ denote its square-free part. Then $V(f) = V(\overline{f})$.*

The proposition shows that we can always assume that a curve is induced by a square-free polynomial. In our algorithms, we will also demand squarefreeness of all input polynomials (or alternatively, we initially compute the square-free part, as discussed in Section 2.4.3).

We consider f as an element of $\mathbb{Z}[x][y]$, and decompose it into $f = \mathrm{cont}(f) \cdot \mathrm{pp}(f)$, with the content $\mathrm{cont}(f) \in \mathbb{Z}[x]$, and the coefficients of the primitive part $\mathrm{pp}(f)$ having no common divisor. For further simplification, we assume that polynomials are primitive, which means that $\mathrm{cont}(f) = 1$. The primitive part of a curve is easy to compute (in comparison with other operations) and the content of bivariate polynomials has a direct geometric meaning.

Definition 2.2.4 (vertical component). An algebraic curve has a vertical component if there exist an $\alpha \in \mathbb{R}$ and an open interval $I \subset \mathbb{R}$ such that $\alpha \times I \subset V(f)$.

Lemma 2.2.5. *If a curve f has a vertical component at α, then it contains the whole vertical line $x = \alpha$ as a component, and $x - \alpha$ is a divisor of the content of f.*

Proof. Consider the polynomial $f_\alpha(y) = f(\alpha, y) \in \mathbb{R}[y]$. If f has a vertical component at α, f_α has infinitely many roots, which is only possible if it is identically zero. Consequently, $f(\alpha, \beta) = 0$ for any $\beta \in \mathbb{R}$, so the vertical line is contained. It follows that the irreducible $(x - \alpha)$ divides f, and since it obviously cannot divide the primitive part, it divides the content. □

Vice versa, it is very easy to see that each real root of the content implies a vertical line component of the curve at this position. Thus, when passing from a curve to its primitive part, we simply remove the vertical components.

The theory of algebraic curves is further simplified if we assume that all curves are irreducible. Indeed, according to (2.1), each curve can be decomposed into finitely many irreducible curves. However, doing so in practice brings up the problem of efficient factorization of a polynomial. Our methods do not demand irreducibility of the polynomials and accordingly, we derive the theory for (square-free and primitive, but) reducible polynomials as well.

Definition 2.2.6 (critical point). A point $p \in \mathbb{R}^2$ is called a *critical point* of an algebraic curve f if $f(p) = \frac{\partial f}{\partial y}(p) = 0$.

Equivalently, the critical points are the set $V(f) \cap V(\frac{\partial f}{\partial y})$. Critical points have a direct geometric meaning (Figure 2.2): The tangent of f at a point p is orthogonal to the *gradient vector* $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$. Thus, if a point is critical, its tangent is either vertical, or not uniquely defined.

Definition 2.2.7 (singular point). A point p is *singular* for a curve f if $f(p) = \frac{\partial f}{\partial x}(p) = \frac{\partial f}{\partial y}(p) = 0$. In this case, we also call p a *singularity*. A point on f that is not singular is called *regular*.

The following famous theorem shows that an algebraic curve can be locally parameterized by a univariate function around regular points. Although this holds true in more general settings, we restrict ourselves to the case of bivariate polynomials. The stated version is proved completely in [For05, §8] and [Kön93, §3.6]. See also [Lan68, §XVII.3] and [Apo74, §13.4] for slightly weaker formulations.

Theorem 2.2.8 (implicit function theorem). *Let f be a polynomial and $p = (x_0, y_0)$. If $\frac{\partial f}{\partial y}(p) \neq 0$, there exist open neighborhoods V around x_0, W around y_0, and a C^∞ function*[11] *$g : V \to W$ such that, for all $x \in V$, $y = g(x)$ is the unique solution in W for $f(x, y) = 0$. Moreover, the derivative of g is given by:*

$$g'(x) = -\frac{\frac{\partial f}{\partial x}(x, g(x))}{\frac{\partial f}{\partial y}(x, g(x))}$$

[11] A C^∞ function is a function that can be continuously differentiated arbitrarily often.

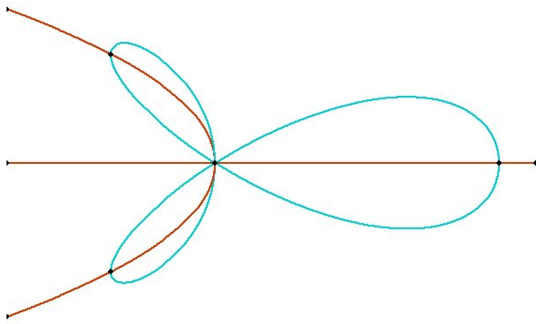


Figure 2.2. A curve (in blue) and its derivative with respect to y (in brown). Note that the intersections are exactly at those positions where the curve has vertical tangent lines, or is singular.

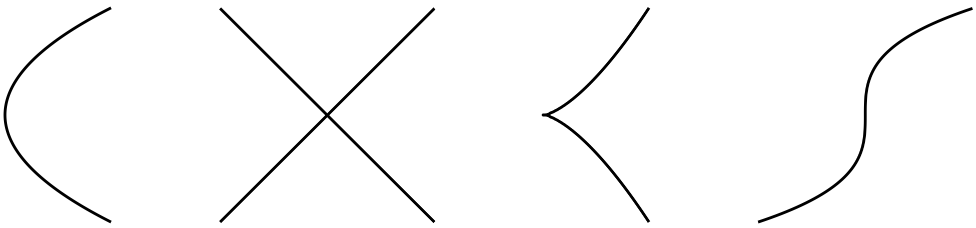


Figure 2.3. Examples of critical points: (left to right) an ordinary x-extreme point, a double point (self-intersection), a cusp, and a vertical inflection point. The first and the last examples are regular points, whereas the self-intersection and the cusp are singular.

The implicit function theorem is the key ingredient for decomposing an algebraic curve into a set of disjoint function graphs, and points in between.

Definition 2.2.9 (strongly critical x-coordinate). A value $x_0 \in \mathbb{R}$ is called a *strongly critical x-coordinate* if $\deg f(x_0, y) < \deg_y f(x, y)$ or if there exist some $y_0 \in \mathbb{R}$ such that (x_0, y_0) is critical.

Theorem 2.2.10 (delineability theorem). *Let $f \in \mathbb{Z}[x, y]$ be square-freeand $I \subset \mathbb{R}$ be an open interval not containing any strongly critical x-coordinate of f. Then, f is delineable over I, which means that there exist C^∞ functions $g_1, \ldots, g_m : I \to \mathbb{R}$ (with $m \geq 0$) such that:*

- *$g_1(x_0) < \ldots < g_m(x_0)$ for all $x_0 \in I$*
- *$V(f) \cap (I \times \mathbb{R}) = \bigcup_{i=1}^{m} \{(x, g_i(x)) \mid x \in I\}$.*

In other words, f decomposes over I into m disjoint function graphs.

For the proof, we need a bound on the size of roots of univariate polynomials. This bound will also come in handy in other contexts.

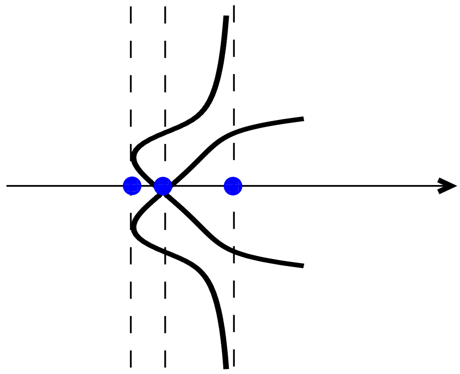


Figure 2.4. The curve $V((x-1)y^4 + y^2 - x^2)$ and its strongly critical x-coordinates. The rightmost one is not caused by a critical point but because the degree drops for $x = 1$. Note that the curve decomposes into disjoint function graphs between two strongly critical x-coordinates.

Theorem 2.2.11. *Let $g = \sum_{i=0}^{n} a_i t^i$ with $a_i \in \mathbb{R}$, and let $\alpha \in \mathbb{C}$ be a root of g. Then,*

$$|\alpha| \leq 1 + \max_{i \neq n}\{\frac{|a_i|}{|a_n|}\}.$$

Proof. Let α be a root of g. Then,

$$|a_n||\alpha|^n \quad \leq \quad \sum_{i=0}^{n-1} |a_i||\alpha|^i \leq \max_{i=0,\ldots,n-1}\{|a_i|\} \sum_{i=0}^{n-1} |\alpha|^i = \max_{i=0,\ldots,n-1}\{|a_i|\} \frac{|\alpha|^n - 1}{|\alpha| - 1}$$

It follows that

$$|\alpha| - 1 \leq \max_{i=0,\ldots,n-1}\{\frac{|a_i|}{|a_n|}\} \underbrace{\frac{|\alpha|^n - 1}{|\alpha|^n}}_{\leq 1}. \quad \square$$

Proof of Theorem 2.2.10. It is enough to prove the statement for any closed interval $J \subset I$. Let $J = [c, d]$ be such a closed interval. Then, $V(f)$ is bounded inside J (i.e., the set $\{y \in \mathbb{R} \mid \exists (x, y) \in V(f) : x \in [c, d]\}$ has a finite supremum and finite infimum) according to Theorem 2.2.11, and so, $V := V(f) \cap (J \times \mathbb{R})$ is a compact set. Since each point in V is regular, applying Theorem 2.2.8 (implicit function theorem) for each point gives function graphs inducing an open covering; we choose a finite subcover $S_1, \ldots, S_M$ of function graphs. By construction $\bigcup_{i=0}^{M} S_i = V$.

The function graphs g_i are now constructed as follows. Start with $q := p_i$, the i-th point in the fiber at c (c is the left endpoint of J). Choose some S of the finite subcover that contains q. Since S is a function graph of a continuous function over an open interval, there is some limit point q' on its right side. Since f, considered as a function $\mathbb{R}^2 \to \mathbb{R}$, is continuous as well, $f(q') = 0$, so q' is contained in another set S' of the subcover. Set $q := q'$ and $S := S'$ and continue the process until it yields a set S that contains a point in the fiber at d (d is the right endpoint of J). This must eventually happen, since the subcover is finite.

Clearly, $g_1, \dots, g_m$ thus obtained are C^∞ functions, and they are disjoint since an intersection would violate the implicit function theorem. Moreover, choosing any point $p \in V$, the same construction as above performed on the left side yields one of the points in the fiber at c on the same function graph. Thus, each point of V is covered by the functions graphs of $g_1, \dots, g_m$. □

2.2.3. Segmentation of algebraic curves

Theorem 2.2.10 proposes a decomposition of the curve into points on fibers of strongly critical x-coordinates and disjoint function graphs in between. This is almost what we are aiming for; however, it will be more convenient for us to work with objects that are closed in $\mathbb{R}^2$. It is not hard to turn the decomposition from Theorem 2.2.10 into a *segmentation* of the curve, defined as follows.

Definition 2.2.12 (x-monotone segment). An *x-monotone segment s* is the image of a continuous function $\varphi : I \to \mathbb{R}^2$ with I the open, half-open, or closed unit interval and such that

- For $t_1 < t_2 \in I$, $\varphi(t_1) \le \varphi(t_2)$ with respect to the lexicographic ordering in $\mathbb{R}^2$. In particular, the segment is x-monotone.
- If $0 \notin I$, then $\lim_{t\to 0} \varphi(t) \notin \mathbb{R}^2$, and if $1 \notin I$, then $\lim_{t\to 1} \varphi(t) \notin \mathbb{R}^2$.

If $0 \in I$, $\varphi(0)$ is the *left endpoint* of the segment, if $1 \in I$, $\varphi(1)$ is the *right endpoint* of the segment.

Definition 2.2.13 (segmentation). For an algebraic curve $V(f)$, let $C = \{c_1, \dots, c_s\}$ be an (ordered) finite set containing all its strongly critical x-coordinates.

The *segmentation* of $V(f)$ with respect to C is a set of x-monotone segments $s_1, \dots, s_p$ which satisfy the following properties

- $s_1 \cup \dots \cup s_p = V(f)$
- Each pair of segments intersects at most at the endpoints
- The x-range of a segment is an open, half-open, or closed interval whose boundaries are either $-\infty$ and c_1, c_s and $+\infty$, or c_i and c_{i+1} for $1 \le i \le s-1$.

A segmentation is cylindrical, which means that, the x-ranges of two segments either are the same or have at most an endpoint in common. See Figure 2.5 for an example. The general existence of such a segmentation follows from the next theorem, which shows that the number of strongly critical x-coordinates is bounded. We shall also quantify the number of required segments.

Theorem 2.2.14. *If f is square-free with $\deg f = n$, there are no more than $n(n-1)$ many strongly critical x-coordinates.*

Proof. For the proof, we need to pass temporarily to the projective space $\mathbb{P}(\mathbb{C})^2$, which consists of points $(a : b : c)$ with either $a, b \in \mathbb{C}$ and $c = 1$, or $a \in \mathbb{C}, b = 1$, and $c = 0$. $\mathbb{C}^2$ is embedded in $\mathbb{P}(\mathbb{C})^2$ via $(a, b) \mapsto (a : b : 1)$. $(a : 1 : 0)$ is usually called a *line at infinity*. For $f = \sum a_{ij} x^i y^j$ of degree n, its homogenization is given by $f^h := \sum a_{ij} x^i y^j z^{n-i-j}$. It is easy to see that $V_{\mathbb{C}}(f) \subset V_{\mathbb{P}(\mathbb{C})}(f^h)$, according to the canonical embedding from above.

By Bezout's theorem (see [Wal50, III.3] [BK81, 6.1] [Gib98, 14.4]), two non-overlapping curves, $V(f)$ and $V(g)$, have exactly $\deg f \cdot \deg g$ common points in $\mathbb{P}(\mathbb{C})^2$. So, $V(f)$ and

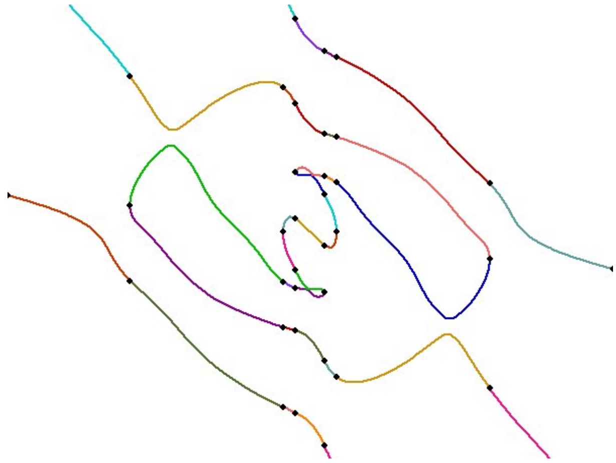


Figure 2.5. The segmentation of an algebraic curve. Each colored part corresponds to a segment, and the black dots are the endpoints.

$V(\frac{\partial f}{\partial y})$ have at least $n(n-1)$ common points. It is left to show that each strongly x-critical coordinate is caused by such a common point.

Recall that there are two cases that cause a strongly critical x-coordinate. Either x_0 has a critical point in its fiber or the y-degree of f drops when setting $x = x_0$. For the first case, if (x_0, y_0) is a critical point, then clearly, $(x_0 : y_0 : 1)$ is a common point of $V(f)$ and $V(\frac{\partial f}{\partial y})$ in the projective plane. For the latter case, note that the y-degree drops if and only if x_0 is a root of the leading coefficient. Then $(x_0 : 1 : 0)$ is a point on the projective curve $V_{\mathbb{P}(\mathbb{C})}(f^h)$. But x_0 is also a root of the leading coefficient of $\frac{\partial f}{\partial y}$, thus, $(x_0 : 1 : 0)$ is again a common point of the curves. $\square$

Corollary 2.2.15. *For an algebraic curve induced by $f \in \mathbb{Z}[x, y]$ with $\deg f = n$, and for any choice C according to Definition 2.2.13, there exists a segmentation with respect to C in no more than n^3 x-monotone segments.*

Proof. Partitioning the x-axis with respect to the strongly critical x-coordinates yields $n(n-1)+1$ open intervals, where f is delineable according to Theorem 2.2.10. Each function graph over such an interval is turned into an x-monotone segment by taking its closure in $\mathbb{R}^2$. In this way, we get up to n x-monotone segments per interval, which yields $n^3 - n^2 + n$ segments.

The union of these segments does not yet cover $V(f)$: there might be isolated points on fibers, that is, with no further (real) point of $V(f)$ in an open neighborhood. For such points, we create degenerated segments whose image consists of a single point (note that this is covered by our definition of x-monotone segments). Since such points are clearly critical, there cannot be more than $n(n-1)$ of them, therefore, the total number of segments is bounded by n^3. $\square$

We have excluded vertical components in this part. However, the segmentation can be extended to such curves in a straightforward way and it is not hard to show that a segmentation with up to n^3 segments also exists for non-primitive curves (consider a segmentation of the primitive part with respect to the strongly critical x-coordinates of $V(f)$, and add vertical segments on the suitable fibers).

We classify the possible segment types that can appear in the segmentation with respect to $C = \{c_1, \ldots, c_s\}$. There are two special segment types that we have already discussed, namely degenerate segments for isolated points and vertical segments. Non-degenerate and non-vertical segments are classified by their behavior at the *curve ends*, that is, their left and right boundaries.

We restrict ourselves to the left boundary. Three types of curve ends can be classified:

1. If the segment has a left endpoint, the segment is called *bounded on the left*.
2. If the x-range of the segment is $(-\infty, c_1]$ or $(-\infty, c_1)$, the segment is called *unbounded in the x-direction (on the left)*.

That leaves the third case where the x-range is bounded by some c_i and c_{i+1}, but the left endpoint does not exist. It follows easily that any sequence of points approaching the left boundary of s must be unbounded in the y-direction (otherwise, some subsequence of points would converge to a finite y-coordinate, and the segment would have to converge to the limit by continuity). Moreover, any such sequence must diverge to the same direction, either to $-\infty$ or to $+\infty$ (otherwise, the segment would cross the x-axis infinitely many times). Thus, the left end of the segment either diverges to $(c_i, -\infty)$ or $(c_i, +\infty)$. We call the segment a *vertically asymptotic arc (at its left end) for $x = c_i$ towards $y = -\infty$*, or *towards $y = +\infty$*, respectively. Indeed, the segment comes arbitrarily close to the vertical line $x = c_i$, but never reaches it.

Lemma 2.2.16. *If there is a vertically asymptotic segment of $V(f)$ for $x = \alpha$ (either at its left or its right end), then α is a root of $\mathrm{lcf}_y(f) \in \mathbb{R}[x]$.*

Proof. Let $a_n(x), \ldots, a_0(x)$ denote, as usual, the coefficients of $f \in \mathbb{Z}[x][y]$. Assume that $a_n(\alpha) \neq 0$. There is an interval I around α where a_n does not vanish. Set

$$\begin{aligned} \alpha_0 &:= \min_{x \in I} |a_n(x)| \\ \alpha_1 &:= \max_{x \in I, j=1,\ldots,n} |a_i(x)| \end{aligned}$$

By Theorem 2.2.11, $1 + \frac{\alpha_1}{\alpha_0}$ is an upper bound for all fiber points in the x-range I, thus, $V(f)$ is bounded in this x-range. This contradicts the fact that a segment diverges to $(\alpha, \pm\infty)$. ☐

Summary

Algebraic curves are defined by the vanishing set of an equation in two variables. Of special interest are their critical points, which means singularities or points with a vertical tangent line. They induce a (cylindrical) segmentation of the curve into x-monotone segments and will form the input segments of our arrangement algorithm.

2.3. (Sub)resultants and Sturm-Habicht sequences

We saw in the last section that critical points play an important role when decomposing an algebraic curve into x-monotone parts. We will next introduce resultants, which are a well-known tool in algebra for projecting the solutions of a polynomial system in a lower-dimensional space. In particular, the resultant of $f \in \mathbb{Z}[x][y]$ and $\frac{\partial f}{\partial y}$ yields a univariate polynomial whose roots contain all strongly x-critical coordinates. Moreover, we discuss subresultants, and the closely related Sturm-Habicht sequences, which generalize resultants and give additional information about the fiber of critical x-coordinates.

2.3.1. Resultants

Consider two polynomials $f, g \in D[t]$, where D is an arbitrary domain. Assume that $n := \deg f \geq \deg g =: m$. We want to answer the question of whether f and g have a common factor, that is, whether $\deg(f, g) \geq 1$. The next well-known theorem reformulates this question in terms of a linear combination of f and g.

Theorem 2.3.1. *$\deg \gcd(f, g) \geq k$ if and only if there are polynomials u and $v \in D[t]$ of degree at most $m-k$ and $n-k$, respectively, such that $uf + vg = 0$.*

Proof. If $h := \gcd(f, g)$ has a degree of k, set $u := \frac{g}{h}$ and $v := -\frac{f}{h}$. For the other direction, assume that u and v exist as required. Then f divides vg. Since v is of degree at most $n-k$, there must be a factor of f of degree at least k that also divides g, which proves that $\gcd(f, g) \geq k$. □

It follows that f and g have a common factor if and only if there are polynomials u and v of degree $m-1$ and $n-1$, respectively, such that $uf + vg = 0$. This can be written as a linear system of equations in the coefficients of f and g, and leads to the following definition.

Definition 2.3.2 (Sylvester matrix). The *Sylvester matrix* of $f = \sum_{i=0}^{n} a_i t^i$ and $g = \sum_{i=0}^{m} b_i t^i$ is the following $(n+m) \times (n+m)$ matrix, composed of m rows of the coefficients of f and n rows of the coefficients of g:

$$\mathrm{Syl}(f,g) = \begin{array}{c} \left(\begin{array}{cccccc} a_n & & \cdots & a_0 & & \\ & \ddots & & & \ddots & \\ & & a_n & \cdots & & a_0 \\ b_m & & \cdots & b_0 & & \\ & \ddots & & & \ddots & \\ & & b_m & \cdots & & b_0 \end{array} \right) \end{array} \begin{array}{l} \left.\vphantom{\begin{array}{c} a \\ \ddots \\ a \end{array}}\right\} m \text{ rows} \\ \left.\vphantom{\begin{array}{c} b \\ \ddots \\ b \end{array}}\right\} n \text{ rows} \end{array}$$

Definition 2.3.3 (resultant). The resultant of f and g is defined as

$$\mathrm{res}(f, g) := \det\left(\mathrm{Syl}(f, g)\right) \in D.$$

With the argument outlined above, we get

Corollary 2.3.4. *The following conditions are equivalent*
- $\mathrm{res}(f, g) = 0$.
- $\deg \gcd(f, g) \geq 1$.
- *There exists a common root α of f and g in $\overline{Q(D)}$, the algebraic closure of the field of fractions of D.*

For the next theorem (see also [Yap00, §4.4, Lemma 4.9] [BPR06, §8.3.5]), note that a homomorphism of domains $\varphi : D \to D'$ induces a homomorphism $D[t] \to D'[t]$ in a natural way by mapping all the coefficients of the polynomial. We also denote the mapping of the polynomial rings by φ.

Theorem 2.3.5 (specialization property). *Let $\varphi : D \to D'$ be a homomorphism of domains with* $\mathrm{lcf}(f), \mathrm{lcf}(g) \notin \ker\varphi$. *Then,*

$$\varphi(\mathrm{res}(f,g)) = \mathrm{res}(\varphi(f), \varphi(g)).$$

Proof. The map φ also induces a mapping of matrices over D to matrices over D' by mapping its coefficients. Since φ preserves the degree of f and g by assumption, it follows that

$$\varphi(\mathrm{Syl}(f,g)) = \mathrm{Syl}(\varphi(f), \varphi(g)).$$

The determinant is a sum of products of the matrix entries, and thus,

$$\varphi(\det \mathrm{Syl}(f,g)) = \det \varphi(\mathrm{Syl}(f,g)).$$

□

The following expression essentially states that the resultant of f and g is the product of the roots of f, evaluated at g (or vice versa). See [BPR06, Thm.4.16] [Yap00, Thm.6.15] for proofs.

Theorem 2.3.6. *Let $f = a_n \prod_{i=1}^{n}(t - \alpha_i)$ and $g = b_m \prod_{j=1}^{m}(t - \beta_j) \in D[t]$. Then,*

$$\begin{aligned} \mathrm{res}(f,g) &= a_n^m b_m^n \prod_{i=1}^{n}\prod_{j=1}^{m}(\alpha_i - \beta_j) \\ &= a_n^m \prod_{i=1}^{n} g(\alpha_i) \\ &= (-1)^{nm} b_m^n \prod_{j=1}^{m} f(\beta_j). \end{aligned}$$

We switch back to the case where f and g are bivariate integer polynomials. By interpreting both as elements of $\mathbb{Z}[x][y]$, we can still talk about their Sylvester matrix, whose entries are in $\mathbb{Z}[x]$, and about their resultant, which we denote by $\mathrm{res}_y(f,g) \in \mathbb{Z}[x]$.

Theorem 2.3.7. *Let $f, g \in \mathbb{Z}[x,y]$ be coprime (i.e., $\deg \gcd(f,g) = 0$), and $r := \mathrm{res}_y(f,g) \in \mathbb{Z}[x]$. Then $r \neq 0$, and the following two statements are equivalent:*
- *α is a root of r.*
- *$\deg f(\alpha, y) < \deg_y f(x,y)$ and $\deg g(\alpha, y) < \deg_y g(x,y)$, or there exist $\beta \in \overline{\mathbb{Q}}$ such that $f(\alpha,\beta) = 0 = g(\alpha,\beta)$.*

Proof. It is clear that $r \neq 0$ because f and g are coprime. Assume that $r(\alpha) = 0$. If $\deg f(\alpha, y) = \deg_y f(x,y)$, and $\deg g(\alpha,y) = \deg_y g(x,y)$, we can apply Theorem 2.3.5 using the substitution homomorphism $x \mapsto \alpha$ and obtain

$$0 = r(\alpha) = \mathrm{res}(f(\alpha,y), g(\alpha,y)).$$

It follows from Corollary 2.3.4 that there exists some $\beta \in \overline{\mathbb{Q}}$ with $f(\alpha,\beta) = g(\alpha,\beta) = 0$.

Vice versa, if $\deg f(\alpha,y) < \deg_y f(x,y)$ and $\deg g(\alpha,y) < \deg_y g(x,y)$, note that α is both a root of $\mathrm{lcf}_y(f) \in \mathbb{Z}[x]$ and of $\mathrm{lcf}_y(g) \in \mathbb{Z}[x]$. Laplace expansion in the first column of the Sylvester matrix yields

$$r := \mathrm{res}_y(f,g) = \mathrm{lcf}_y(f) \cdot U + \mathrm{lcf}_y(g) \cdot V$$

with some $U, V \in \mathbb{Z}[x]$. It follows that $r(\alpha) = 0$.

It remains the case that there exist $\beta \in \overline{\mathbb{Q}}$ such that $f(\alpha, \beta) = 0 = g(\alpha, \beta)$. If neither f nor g drop their degree at α, $r(\alpha) = 0$ again follows by the specialization property and Corollary 2.3.4. However, it remains the case that exactly one of the polynomials drops its degree. Note that $\operatorname{res}(f, g) = \pm\operatorname{res}(g, f)$, so w. l. o. g., assume that $\operatorname{lcf}_y(g)(\alpha) = 0$, and $\operatorname{lcf}_y(f)(\alpha) \neq 0$. Let k be the degree of $g(\alpha, y)$. It follows that

$$r(\alpha) = (\det \operatorname{Syl}(f, g))(\alpha) = \det(\operatorname{Syl}(f, g)(\alpha)),$$

where $\operatorname{Syl}(f, g)(\alpha)$ means we apply the substitution homomorphism $x \to \alpha$ to any entry of $\operatorname{Syl}(f, g)$. This makes the first $m - k$ entries of each row with g-coefficients vanish:

$$
\begin{aligned}
r(\alpha) \quad &= \left|\begin{array}{cccccccc}
a_n(\alpha) & & & \cdots & & a_0(\alpha) & & \\
& \ddots & & & & & \ddots & \\
& & a_n(\alpha) & & \cdots & & & a_0(\alpha) \\
0 & \cdots & b_k(\alpha) & \cdots & & b_0(\alpha) & & \\
& \ddots & & \ddots & & & \ddots & \\
& & 0 & \cdots & b_k(\alpha) & \cdots & & b_0(\alpha)
\end{array}\right| \\
&= a_n(\alpha)^{m-k} \cdot \left|\begin{array}{cccccc}
a_n(\alpha) & & \cdots & a_0(\alpha) & & \\
& \ddots & & & \ddots & \\
& & a_n(\alpha) & \cdots & & a_0(\alpha) \\
b_k(\alpha) & & \cdots & b_0(\alpha) & & \\
& \ddots & & & \ddots & \\
& & b_m(\alpha) & \cdots & & b_0(\alpha)
\end{array}\right|
\begin{array}{l}
\left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} k \text{ rows} \\
\left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} n \text{ rows}
\end{array} \\
&= a_n(\alpha)^{m-k} \cdot \operatorname{res}(f(\alpha, y), g(\alpha, y)).
\end{aligned}
$$

By Corollary 2.3.4, it follows that the second factor vanishes, and therefore, $r(\alpha) = 0$. ☐

We can immediately conclude that strongly critical x-coordinates are roots of a resultant polynomial:

Corollary 2.3.8. *Let α be a strongly critical x-coordinate of a curve $V(f)$, with f a square-free polynomial. Then, α is a root of $\operatorname{res}_y(f, \frac{\partial f}{\partial y})$.*

Proof. First of all, note that $\operatorname{res}_y(f, \frac{\partial f}{\partial y}) \neq 0$ since f is square-free. Recall the definition of strongly critical (Definition 2.2.9): If $\deg f(\alpha, y) < \deg_y f(x, y)$, also $\deg \frac{\partial f}{\partial y}(\alpha, y) < \deg_y \frac{\partial f}{\partial y}(x, y)$, and the resultant vanishes. Otherwise, if there exists a critical point (α, β), $f(\alpha, y)$ and $\frac{\partial f}{\partial y}(\alpha, y)$ have a common root at β, so the resultant vanishes as well. ☐

Note that, conversely, not every root of $\operatorname{res}_y(f, \frac{\partial f}{\partial y})$ is a strongly critical x-coordinate. The reason is that, according to Theorem 2.3.7, the common root of $f(\alpha, y)$ and $g(\alpha, y)$ might also be non-real. In other words, the complex algebraic curves $V_{\mathbb{C}}(f)$ and $V_{\mathbb{C}}(g)$ intersect at a non-real point whose x-coordinate happens to be real.

Definition 2.3.9 (critical x-coordinate). Let $f \in \mathbb{Z}[x, y]$ be square-free, and $\alpha \in \mathbb{R}$. We call α a *critical x-coordinate* if α is a root of $\operatorname{res}_y(f, \frac{\partial f}{\partial y})$.

The considerations above demonstrate the usefulness of resultants for our computational purposes: They encode the x-coordinates of intersection points of two algebraic curves, as well as the x-coordinates of critical points of an algebraic curve. Computing the real roots of resultants, and of univariate polynomials in general, is the subject of Section 2.4.4. Next, we study a generalization of the resultant that allows us to deduce additional information about the fiber at critical x-coordinates.

2.3.2. Subresultants and the gcd

Recall the result of Theorem 2.3.1. So far, we have only applied it for the case where $k = 1$. However, introducing the slightly more general *subresultants*, we can not only identify critical x-coordinates, but also determine the degree of the gcd at these positions, and even compute the gcd itself.

Definition 2.3.10 (Sylvester submatrix). The *i-th Sylvester submatrix* of f and g is defined to be the following $(n + m - 2i) \times (n + m - i)$ matrix, composed of $m - i$ rows of the coefficients of f and $n - i$ rows of the coefficients of g

$$\mathrm{Syl}_i(f, g) = \left(\begin{array}{ccccc} a_n & & \dots & a_0 & \\ & \ddots & & & \ddots & \\ & & a_n & \dots & & a_0 \\ b_m & & \dots & b_0 & \\ & \ddots & & & \ddots & \\ & & b_m & \dots & & b_0 \end{array}\right) \begin{array}{l} \left.\vphantom{\begin{array}{c} a\\ \ddots \\ a \end{array}}\right\} m - i \text{ rows} \\ \left.\vphantom{\begin{array}{c} b\\ \ddots \\ b \end{array}}\right\} n - i \text{ rows} \end{array}$$

$\mathrm{Syl}_0(f, g)$ coincides with the Sylvester matrix $\mathrm{Syl}(f, g)$. The subresultants of f and g are defined by certain minors of the Sylvester submatrices.

Definition 2.3.11 ($M_{k,t}$ minors). For any $p \times q$-matrix B with $p \leq q$, we define $M_{k,t}(B)$ with $1 \leq k \leq p$ and $0 \leq t \leq q - k$ to be the $k \times k$ minor of B extracted from the first k rows of B, the first $k - 1$ columns of B, and the $(k + t)$-th column of B. $M_{k,0}(B)$ is called the *k-th leading principal minor of B*.

Definition 2.3.12 (subresultants). For $i \in \{0, \dots, m - 1\}$, set $S_i := \mathrm{Syl}_i(f, g)$. The subresultants of f and g are build out of minors of S_i as follows:

1. The *i-th (polynomial) subresultant* of f and g is defined as the polynomial

$$\mathrm{Sres}_i(f, g) := \sum_{j=0}^{i} M_{n+m-2i,i-j}(S_i) \cdot t^j \in D[t].$$

2. The *i-th principal subresultant coefficient (psc)* is given by

$$\mathrm{sres}_i(f, g) := \mathrm{coef}_i(\mathrm{Sres}_i(f, g)) = M_{n+m-2i,0}(S).$$

Different from this notation, some authors blur the difference between polynomial and principal subresultants, and call the psc's subresultants. See [vzGL03] for an overview of how the term "subresultants" is used in the literature.

Note that $\mathrm{Syl}_i(f,g)$ can be interpreted as a multiplication matrix. With polynomials u and v of degree $m-i-1$ and $n-i-1$, respectively, we can associate a coefficient vector $(u_{m-i-1},\ldots,u_0,v_{n-i-1},\ldots,v_0)$ of dimension $(n+m-2i)$. $\mathrm{Syl}_i(f,g)$ can then be interpreted as the matrix associated with the map $(u,v)\mapsto uf+vg$. By considering the first $(n+m-2i)$ rows and columns, we obtain:

Lemma 2.3.13. $\mathrm{sres}_i(f,g) = 0$ *if and only if there exist non-zero* $u_i, v_i \in D[x]$ *with* $\deg(u_i)\le m-i-1$, $\deg(v_i)\le n-i-1$ *and* $\deg(u_i f + v_i g) < i$.

Consequently, the psc's determine the degree of the gcd of f and g with the following property [BPR06, Prop.4.24]:

Lemma 2.3.14.

$$\deg(\gcd(f,g)) = \min\{k\in\{0,\ldots,m-1\} \mid \mathrm{sres}_k(f,g)\neq 0\}$$

and $\deg(\gcd(f,g)) = m$ *if all psc's are zero (this happens if and only if* g *divides* f*).*

Proof. The statement is equivalent to

$$\deg\gcd(f,g)\ge k \text{ if and only if } \mathrm{sres}_0(f,g)=\ldots=\mathrm{sres}_{k-1}(f,g)=0.$$

Assume that $h := \gcd(f,g)$ is of degree at least k. Set $u := \frac{g}{h}$ and $v := -\frac{f}{h}$. Then $u_i f + v_i g = 0$, and $\deg u \le m-k$ and $\deg v\le n-k$. It follows that the right-hand side of Lemma 2.3.13 is satisfied for $i=0,\ldots,k-1$. Thus, $\mathrm{sres}_0(f,g)=\ldots=\mathrm{sres}_{k-1}(f,g)$.

For the opposite direction, we use induction on k. For $k=1$, if $\mathrm{sres}_0(f,g)=\mathrm{res}(f,g)=0$, then $\deg\gcd(f,g)\ge 1$ by Corollary 2.3.4. Assume for any $k>1$ that $\mathrm{sres}_0(f,g)=\ldots=\mathrm{sres}_{k-1}(f,g)=0$. By induction hypothesis, $\deg(\gcd(f,g))\ge k-1$. Since $\mathrm{sres}_{k-1}(f,g)=0$, Lemma 2.3.13 implies the existence of u_{k-1} and v_{k-1} of degrees $m-k$ and $n-k$, respectively, such that $\deg(u_{k-1}f+v_{k-1}g)<k-1$. Since the gcd of f and g also divides $u_{k-1}f+v_{k-1}g$ and is of a degree of at least $k-1$, it follows that $u_{k-1}f+v_{k-1}g=0$. Hence, $u_{k-1}f=-v_{k-1}g$ is a common multiple of f and g, of degree at most $n+m-k$. It follows that the gcd of f and g has a degree of at least k. ☐

Another well-known way to define subresultants is by the determinant.

$$\mathrm{Sres}_i(f,g) = \left|\begin{array}{ccccc} a_n & \cdots & \cdots & a_{2i-m+2} & t^{m-i-1}f \\ & \ddots & & \vdots & \vdots \\ & a_n & \cdots & a_{i+1} & f \\ b_m & \cdots & \cdots & b_{2i-n+2} & t^{n-i-1}g \\ & \ddots & & \vdots & \vdots \\ & b_m & \cdots & b_{i+1} & g \end{array}\right| \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\ \ddots\\ a\end{array}}\right\} m-i \text{ rows} \\ \left.\vphantom{\begin{array}{c}a\\ \ddots\\ a\end{array}}\right\} n-i \text{ rows}\end{array} \tag{2.3}$$

with $a_j = 0 = b_j$ for negative indices j. The equivalence of both definitions is proven by exploiting the linearity of the determinant in the last column (e.g., [BT71]).

On the other hand, Laplace expansion on the last column in (2.3) yields

$$\mathrm{Sres}_i(f,g) = u_i f + v_i g, \quad \text{where } \deg(u_i)\le m-i-1, \deg(v_i)\le n-i-1. \tag{2.4}$$

Set $d := \deg(\gcd(f,g))$. From Lemma 2.3.14, $\mathrm{Sres}_d(f,g)$ is a polynomial of degree d and by (2.4) it follows that $\mathrm{Sres}_d(f,g)$ is a gcd of f and g.

The cofactors u_i and v_i in (2.4) can also be expressed as determinant of certain matrices.

Definition 2.3.15 (subresultant cofactors). For $i \in \{0, \dots, m\}$, define the *subresultant cofactors* as

$$u_i(f,g) := \begin{vmatrix} a_n & \dots & \dots & a_{2i-m+2} & t^{m-i-1} \\ & \ddots & & \vdots & \vdots \\ & a_n & \dots & a_{i+1} & 1 \\ b_m & \dots & \dots & b_{2i-n+2} & 0 \\ & \ddots & & \vdots & \vdots \\ & b_m & \dots & b_{i+1} & 0 \end{vmatrix} \quad v_i(f,g) := \begin{vmatrix} a_n & \dots & \dots & a_{2i-m+2} & 0 \\ & \ddots & & \vdots & \vdots \\ & a_n & \dots & a_{i+1} & 0 \\ b_m & \dots & \dots & b_{2i-n+2} & t^{n-i-1} \\ & \ddots & & \vdots & \vdots \\ & b_m & \dots & b_{i+1} & 1 \end{vmatrix}.$$

Clearly, $\deg u_i \leq m-i-1$ and $\deg v_i \leq n-i-1$.

The following simple lemma (compare [GVN02]) will play an important role in our algorithm. If f and g share exactly one common root, this root can be expressed by a rational expression in terms of subresultants.

Lemma 2.3.16. *Let $f, g \in \mathbb{R}[t]$. If f and g have exactly one common root β, which is of multiplicity k, then $\beta \in \mathbb{R}$, and, moreover,*

$$\beta = -\frac{\mathrm{coef}_{k-1}(\mathrm{Sres}_k(f,g))}{k \cdot \mathrm{coef}_k(\mathrm{Sres}_k(f,g))}.$$

Proof. By assumption, $\gcd(f,g) = (t-\beta)^k$. Since $\gcd(f,g) \in \mathbb{R}[t]$, it follows that $\beta \in \mathbb{R}$. Moreover, from the above considerations, it is $\mathrm{Sres}_k(f,g) = \mathrm{sres}_k(f,g)(t-\beta)^k$. By comparing the coefficient of t^{k-1}, we obtain

$$\mathrm{coef}_{k-1}(\mathrm{Sres}_k(f,g)) = -k \cdot \mathrm{sres}_k(f,g)\beta.$$

□

The analogue of Theorem 2.3.5 also applies to subresultants (with the same proof):

Theorem 2.3.17 (specialization property for subresultants). *Let $\varphi : D \to D'$ be a homomorphism of domains with $\mathrm{lcf}(f)$ and $\mathrm{lcf}(g) \notin \ker \varphi$. Then,*

$$\varphi(\mathrm{Sres}_i(f,g)) = \mathrm{Sres}_i(\varphi(f), \varphi(g)).$$

In particular, for bivariate polynomials $f, g \in \mathbb{Z}[x][y]$, we can obtain $\mathrm{Sres}_i(f(\alpha,y), g(\alpha,y))$ by plugging α into $\mathrm{Sres}_i(f,g)$, if α is neither a root of the leading coefficient of f nor of g, (both considered to be polynomials in y).

Another important property of subresultants is the so-called *structure theorem*. In the form that we present it, it appears for instance in [BPR06, Theorem 8.30] and [ADTGV04], and goes back to Lickteig and Roy, according to a remark in [Duc00].

Let $f, g \in D[t]$. For shorter notation, we write Sres_i for $\mathrm{Sres}_i(f,g)$, and $\mathrm{lcfS}_k := \mathrm{lcf}(\mathrm{Sres}_k)$.

Theorem 2.3.18 (structure theorem). *Let Sres_d be of degree d and Sres_{d-1} be of degree $e \in \{0, \dots, d-1\}$. Let $\delta := d-1-e$. Then,*

$$\begin{aligned} \mathrm{Sres}_{d-2} &= \dots = \mathrm{Sres}_{e+1} = 0 \\ \mathrm{Sres}_e &= \frac{\mathrm{lcfS}_{d-1}^{\delta} \mathrm{Sres}_{d-1}}{\mathrm{lcfS}_d^{\delta}} \\ \mathrm{Sres}_{e-1} &= \frac{\mathrm{rem}(\mathrm{lcfS}_e \cdot \mathrm{lcfS}_{d-1} \mathrm{Sres}_d, \mathrm{Sres}_{d-1})}{(-1)^{\delta} \mathrm{lcfS}_d^2}. \end{aligned}$$

In particular, all divisions are exact over D.

Theorem 2.3.18 is an improved version of the classical structure theorem discovered by Collins [Col67] and Brown and Traub [BT71]. In this form, it also appears in [Loo82b].

With the structure theorem, the close relation between subresultants and polynomial remainder sequences is established.

Definition 2.3.19 (polynomial remainder sequences (prs)). For $f, g \in D[t]$, a *polynomial remainder sequence* of f and g is a sequence of polynomials $S_1, \ldots, S_k$ with $S_1 = f$ and $S_2 = g$ such that

$$a_i S_{i-1} = Q_i S_i + b_i S_{i+1}$$

with $a_i \in D$, $b_i \in D$, $Q_i \in D[t]$, and $\deg S_{i+1} < \deg S_i$ for $i = 2, \ldots, k-1$, and

$$a_k S_{k-1} = Q_k S_k$$

for $a_k \in D$, $Q_k \in D[t]$.

Definition 2.3.20 (regular and defective subresultants). We call $\mathrm{Sres}_i(f, g)$ *regular*, if its degree is i, and *defective* otherwise.

Corollary 2.3.21. *The sequence of polynomials starting with f and g followed by the regular subresultants forms a polynomial remainder sequence.*

This property of subresultants allows the application of fast methods for their computation (Section 2.4.2). Furthermore, it allows counting the number of real roots of a polynomial, as we describe next.

2.3.3. Real root counting

In this section, we define the *Sturm-Habicht sequence* of f which equals the subresultant polynomials of f and its derivative f', up to certain sign changes. Sturm-Habicht sequences and their relation to real root counting have been studied in [GVRLR98]. In [BPR06], the *signed subresultant sequence* is introduced for the same purpose.[12]

Definition 2.3.22 (Sturm-Habicht sequence). Let $\delta_k := (-1)^{k(k+1)/2}$. For f as above and $k \in \{0, \ldots, n\}$, the *k-th Sturm-Habicht polynomial* of f is defined as

$$\mathrm{StHa}_k(f) = \begin{cases} f & \text{if } k = n \\ f' & \text{if } k = n-1 \\ \delta_{n-k-1}\mathrm{Sres}_k(f, f') & \text{if } 0 \le k \le n-2 \end{cases}$$

The *k-th principal Sturm-Habicht coefficient* is defined as

$$\mathrm{stha}_k(f) := \begin{cases} 1 & \text{if } k = n \\ \mathrm{coef}_k(\mathrm{StHa}_k(f)) & \text{if } k = 0, \ldots, n-1 \end{cases}$$

[12] Both treatments consider the Sturm-Habicht sequence of f and g, which resembles a subresultant sequence of f and $f'g$. We only look at the special case of $g = 1$ here.

We offer a motivation for the preceding definition. Assume for the moment that $f \in \mathbb{R}[t]$. Recall from the last section that the regular subresultants form a polynomial remainder sequence. With *Sturm's Theorem* (Theorem 2.3.25), one can count the number of real roots of f inside any interval.

Definition 2.3.23 (Sturm sequence). A polynomial remainder sequence of f and f' is called a *Sturm sequence* of f if, in any relation

$$a_i S_{i-1} = Q_i S_i + b_i S_{i+1},$$

it holds that $\operatorname{sign}(a_i b_i) < 0$.

Definition 2.3.24 (sign variations). For a sequence $A := [a_0, \ldots, a_n]$ with $a_i \in \mathbb{R}$, the number of sign variations, $\operatorname{Var}(A)$, is the number of pairs (a_i, a_j) with $i < j$ such that $a_i a_j < 0$ and $a_{i+1} = \ldots = a_{j-1} = 0$.

Theorem 2.3.25 (Sturm's theorem). *Let $S_1, \ldots, S_k$ be a Sturm sequence of f, and $a, b \in \mathbb{R}$ with $a < b$. Then, the number of distinct real roots inside $[a, b]$ is given by*

$$\operatorname{Var}(S_1(a), \ldots, S_k(a)) - \operatorname{Var}(S_1(b), \ldots, S_k(b)).$$

For proofs, see [Yap00, §7.2] or [BPR06, Thm. 2.50]. Since we are interested in the total number of real roots, we essentially want to set $a = -\infty$ and $b = +\infty$. In this case, it is enough to consider only the leading coefficients of the Sturm sequence.

Corollary 2.3.26. *Let $S_1, \ldots, S_k$ be a Sturm sequence of f and let $s_1, \ldots, s_k$ be their leading coefficients. The total number of real roots of f is*

$$\sum_{i=1}^{k-1} \varepsilon_i, \quad \textit{with} \quad \varepsilon_i = \begin{cases} 0 & \textit{if } \deg(S_{i+1}) - \deg(S_i) \textit{ is even} \\ \operatorname{sign}(s_{i+1} s_i) & \textit{if } \deg(S_{i+1}) - \deg(S_i) \textit{ is odd} \end{cases}$$

Proof. The sign of $S_i(x)$ for $x \to \pm\infty$ is determined by the sign of its leading coefficient. So, we can choose $[a, b]$ so large that it contains all real roots of f, and additionally

$$\operatorname{sign}(S_i(b)) = s_i, \quad \operatorname{sign}(S_i(a)) = (-1)^{\deg S_i} s_i.$$

According to Theorem 2.3.25, it holds that

$$\#V_{\mathbb{R}}(f) = \operatorname{Var}(\underbrace{(-1)^{\deg S_1} s_1}_{=:t_1}, \ldots, \underbrace{(-1)^{\deg S_k} s_k}_{=:t_k}) - \operatorname{Var}(s_1, \ldots, s_k).$$

Consider two consecutive elements s_i and s_{i+1} and the corresponding t_i and t_{i+1}. If the degree difference is even, then there is a sign change from s_i to s_{i+1} if and only if there is a sign change from t_i to t_{i+1}, thus the difference cancels out, and ε_i is zero accordingly. If the degree difference is odd, then there is a sign change form s_i to s_{i+1} if and only if there is no sign change from t_i to t_{i+1}. Thus, a sign change from s_i to s_{i+1} counts as $+1$ in the difference, and a non-sign-change from s_i to s_{i+1} counts as -1. □

A Sturm sequence for f is thus enough to count the number of real roots of f. But does the Sturm-Habicht sequence of f always yield a Sturm sequence? The answer is no, except in regular cases.

Lemma 2.3.27. *If all Sturm-Habicht polynomials are regular, this means* $\deg \mathrm{StHa}_i(f) = i$ *for all* $i = 0, \ldots, n$*, then* $\mathrm{StHa}_n(f), \ldots, \mathrm{StHa}_0(f)$ *is a Sturm sequence.*

Proof. We have already argued that the corresponding subresultant sequence forms a polynomial remainder sequence, and by Theorem 2.3.18 (structure theorem), it holds that

$$\mathrm{sres}_d(f, f')^2 \mathrm{Sres}_{d-2}(f, f') = Q \cdot \mathrm{Sres}_{d-1}(f, f') + \mathrm{sres}_{d-1}(f, f')^2 \mathrm{Sres}_d(f, f').$$

Note that $e = d - 1$ and $\delta = 0$ since all subresultants are regular. We can rewrite this equality using the Sturm-Habicht polynomials:

$$\underbrace{(\delta_{n-d-1}\mathrm{stha}_d(f))^2 \delta_{n-d+1}}_{=:a} \mathrm{StHa}_{d-2}(f) = Q \cdot \delta_{n-d} \mathrm{StHa}_{d-1}(f) + \underbrace{(\delta_{n-d}\mathrm{stha}_{d-1}(f))^2 \delta_{n-d-1}}_{=:b} \mathrm{StHa}_d(f).$$

To check the Sturm property, we evaluate the sign of $a \cdot b$. Clearly, the quadratic factors can be left out, so this product reduces to $\delta_{n-d+1}\delta_{n-d-1}$, and it is easily verified that $\delta_{\ell+1}\delta_{\ell-1} = -1$ for all $\ell \in \mathbb{Z}$. □

As we have remarked, the regular Sturm-Habicht polynomials do not give a Sturm sequence in general. Still, one can generalize Corollary 2.3.26 to count the total number of real roots using the Sturm-Habicht sequence, even in presence of defective Sturm-Habicht polynomials. The proof of this property is more involved [GVRLR98] [BPR06].

Definition 2.3.28 (root counting function). For a sequence $I := (a_0, \ldots, a_n)$ of real numbers with $a_0 \neq 0$, define

$$C(I) = \sum_{i=1}^{s} \varepsilon_i$$

where s is the number of subsequences of I of the form

$$(a, \underbrace{0, \ldots, 0}_{k}, b)$$

with $a \neq 0, b \neq 0, k \geq 0$. For the i-th subsequence of I, define

$$\varepsilon_i := \begin{cases} 0 & \text{if } k \text{ is odd,} \\ (-1)^{k/2}\mathrm{sign}(ab) & \text{if } k \text{ is even.} \end{cases}$$

Theorem 2.3.29. *For* $f \in \mathbb{R}[x]$ *with* $\deg f = n$*, we have:*

$$C(\mathrm{stha}_n(f), \ldots, \mathrm{stha}_0(f)) = \#\{\alpha \in \mathbb{R} \mid f(\alpha) = 0\}.$$

In other words, the signs of the principal Sturm-Habicht coefficients determine the number of distinct real roots of f.

By definition, the specialization property, as stated in Theorem 2.3.17, also holds for Sturm-Habicht polynomials. Thus, by computing the principal Sturm-Habicht coefficients of a bivariate polynomial $f \in \mathbb{R}[x][y]$, it is possible to determine the number of real roots in any fiber $f(\alpha, y)$ by specializing the sequence for $x = \alpha$. Interestingly, there is no need to compute the full Sturm-Habicht sequence for that purpose; its principal coefficients suffice. However, the most efficient known method to compute those principal coefficients is by computing the whole Sturm-Habicht sequence (compare Section **??**), thus, this fact is currently not exploited.

2.3.4. The intersection multiplicity of two curves

We briefly discuss the concept of intersection multiplicity of two curves $V(f)$ and $V(g)$, and relate it to the resultant of f and g. For that, we consider the shear of a curve, which we have already introduced in Section 2.1. It is nothing but a linear change of coordinates, and corresponds to choosing a new vertical direction.

Definition 2.3.30 (shear transformation). For $s \in \mathbb{Z}$, the *shear* of a curve with *shear factor* s is defined as

$$\mathrm{Sh}_s f := f(x + sy, y).$$

If the shear factor is clear from the context, we omit it and only write $\mathrm{Sh} f$.

Considering the map

$$\mathrm{Sh}_s : \mathbb{R}^2 \to \mathbb{R}^2, (x, y) \mapsto (x - sy, y),$$

one immediately observes that $\mathrm{Sh}_s V(f) = V(\mathrm{Sh}_s f)$. Thus, two points p and q on $\mathrm{Sh}_s f$ are covertical if their preimages p' and q' are lying on the same line of slope $\frac{1}{s}$. In other words, the vector $(s, 1)$ in the original system is chosen as the vertical direction in the transformed coordinate system.

Lemma 2.3.31. *Let $V(f)$ and $V(g)$ be two curves without common components. Then, there exists an $s \in \mathbb{Z}$ such that $V(\mathrm{Sh}_s(f))$ and $V(\mathrm{Sh}_s(g))$ do not have vertically asymptotic arcs, and each intersection point of the curves has a different x-coordinate.*

Proof. Consider $\mathrm{Sh}_s(f)$ as a polynomial in $\mathbb{Z}[s, x, y]$. It follows directly that $\mathrm{lcf}_y(\mathrm{Sh}_s(f)) \in \mathbb{Z}[s]$, and if s is chosen so that $\mathrm{lcf}_y(\mathrm{Sh}_s(f))$ does not vanish, the curve does not have vertically asymptotic arcs, according to Lemma 2.2.16. To exclude the presence of covertical intersection points, consider all intersections of f and g. Clearly, sheared intersection points under $\mathrm{Sh}_s(\cdot)$ are the intersection points of the sheared curve. Consider all lines that pass through at least two intersection points. If s is chosen such that none of these lines becomes vertical, all intersection points have a different x-coordinate (Figure 2.6). After all, we have shown that only finitely many shear factors s violate the demanded properties. $\square$

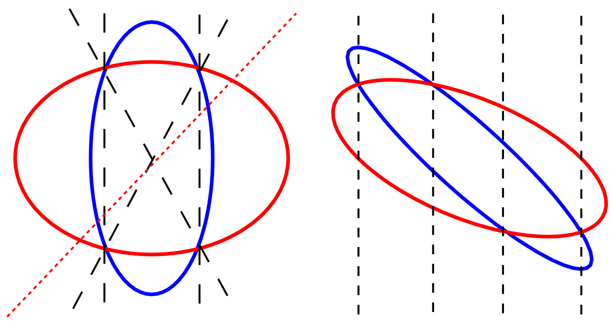


Figure 2.6. Illustration of Lemma 2.3.31. On the right, the sheared curves with $s = 1$.

Curves that satisfy the above properties are sometimes said to be *in generic position*. We will use this term later for a more restrictive set of conditions, and therefore avoid to use it in the current context. Roughly speaking, the intersection multiplicity of p is then the multiplicity of the resultant at p in an appropriate direction.

Definition 2.3.32 (intersection multiplicity). For two curves f and g, let s be a shear factor as in Lemma 2.3.31. Set

$$R := \mathrm{res}_y(\mathrm{Sh}_s(f), \mathrm{Sh}_s(g)).$$

Let $p \in \mathbb{C}^2$ be an intersection point of $V(f)$ and $V(g)$. Let α denote the x-coordinate of its sheared image $\mathrm{Sh}_s(p)$. The *intersection multiplicity* at p is then defined as

$$\mathrm{mult}(p; f, g) := \mathrm{mult}(\alpha, R).$$

For all non-intersection points, we set the intersection multiplicity to zero.

It can be shown that this definition is independent of the actual choice of s (as long as it satisfies Lemma 2.3.31). This is non-trivial to show and requires deeper concepts of algebraic curves which are beyond the scope of this work. We refer the reader to Walker [Wal50, §5], where the intersection multiplicity is defined differently, and the equivalence to our definition is shown. A direct consequence of the given definition is also stated in [Wal50, §5].

Lemma 2.3.33. *Let $\alpha \in \mathbb{R}$ such that $\deg f(\alpha, y) = \deg_y(f)$ and $\deg g(\alpha, y) = \deg_y(g)$. Set $R := \mathrm{res}_y(f, g)$. Then $\mathrm{mult}(\alpha, R)$ equals the sum of all intersection multiplicities in the complex fiber at α. In particular, if there is only one intersection point in the fiber, its multiplicity is $\mathrm{mult}(\alpha, R)$.*

We are mainly interested in the intersection multiplicity at non-critical points of both $V(f)$ and $V(g)$. By Theorem 2.2.8 (implicit function theorem), f can be written in the form $y = \tilde{f}(x)$ around such a point $p = (x_0, y_0)$, and g can be written as $y = \tilde{g}(x)$.

Lemma 2.3.34. *$\mathrm{mult}(p; f, g)$ equals the minimal index k, such that*

$$\tilde{f}^{(k)}(x_0) \neq \tilde{g}^{(k)}(x_0)$$

Indeed, this result matches well the intuition behind intersection multiplicity: a simple intersection point should be a point such that $V(f)$ and $V(g)$ have different slopes at p, and the multiplicity should be higher, the more derivatives coincide. In [Wal50], a generalization of this is taken as the definition of the intersection multiplicity. A third (and also equivalent) definition can be found in [BPR06, p. 148], where the intersection multiplicity is defined as the dimension of the *localization* at the intersection point.

Summary

The resultant of two bivariate polynomials is a suitable tool to represent the x-coordinates of intersection points of the two induced algebraic curves. In particular, the resultant of f and its derivative $\frac{\partial f}{\partial y}$ contains the strongly critical x-coordinates of $V(f)$ as its roots. The subresultants and Sturm-Habicht polynomials reveal further information about polynomials of the form $f(\alpha, y)$, such as the degree of the gcd of two polynomials or the number of real roots, counted without multiplicity. Moreover, after a possible change of coordinates, resultants reveal the multiplicities of intersection points of two curves.

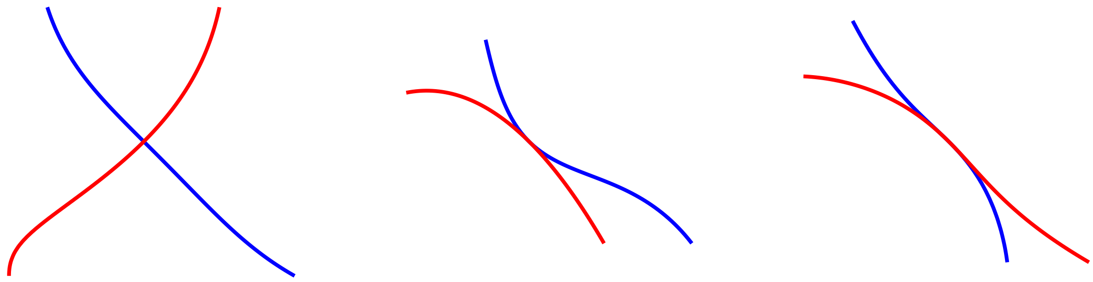


Figure 2.7. Intersections of multiplicity one, two, and three (left to right)

2.4. Computation with polynomials

All information about an algebraic curve is contained in its defining bivariate polynomial. Also, the critical x-coordinates of a curve with respect to one of its variables are encoded in a univariate polynomial, namely the resultant of f and its derivative. It is not surprising that our algorithms to deal with algebraic objects need a basic toolbox of operations dealing with uni- and multivariate polynomials. In this section, we will introduce these methods and analyze their complexity.

For the complexity analysis of algorithms manipulating algebraic objects (such as polynomials over a domain D), the simplest measure is the **arithmetic complexity**, where one just counts the number of operations in D. However, if $D = \mathbb{Z}$, this measure does not take into account the possible coefficient growth during the execution. Therefore, a more refined complexity measure is the **bit complexity**, which counts the maximal number of bit operations performed in the algorithm. Obviously, this gives more meaningful statements about the quality of algorithms dealing with integer polynomials.

Definition 2.4.1 (bitsize).

- The *bitsize* of an integer $n \neq 0$, $\mathrm{Bit}(n)$, is the number of bits needed to represent n in the standard binary expansion. We set $\mathrm{Bit}(n) := \lceil \log |n| \rceil$. Moreover, we set $\mathrm{Bit}(0) = 1$.
- The bitsize $\mathrm{Bit}(r)$ of a rational number $r = \frac{a}{b}$ with a, b coprime is the number of bits needed to represent r. We set $\mathrm{Bit}(r) = \mathrm{Bit}(a) + \mathrm{Bit}(b) = \Theta(\max\{\log |a|, \log |b|\})$

Adding two integers of bitsize n obviously requires $n + 1$ bit operations. For the multiplication, this depends on the chosen algorithm.

Definition 2.4.2 (cost of multiplication). For two integers of bitsize n, let $M(n)$ denote the bit complexity of a multiplication or division.

$M(n) = O(n^2)$ for classic (school) multiplication, $M(n) = O(n^{\log(3)})$ for Karatsuba multiplication [KO63], $M(n) = O(n \log n \log\log n)$ for Schönhage-Strassen multiplication [SS71], and $M(n) = O(n \log n 2^{O(\log^* n)})$ using Fürer's approach [Für07]. In any case, $M(n) = \Omega(n)$, and so we have

$$M(n_1) + M(n_2) \leq M(n_1 + n_2).$$

The standard way to obtain the bit complexity of an algorithm is to analyze the arithmetic

Algorithm 2.1. Univariate addition

INPUT: $f, g \in \mathbb{Z}[t]$ of magnitude (n, τ)
OUTPUT: $f + g$

procedure $\textsc{Add}(f, g)$
  for $j \in \{0, \ldots, n\}$ **do**
    $c_j \leftarrow \text{coef}_j(f) + \text{coef}_j(g)$
  end for
  return $\sum_{j=0}^{n} c_j t^j$
end procedure

complexity A and to bound the maximal size s of integers that occur in the algorithm. Then, the bit complexity of the algorithm can simply be bounded by $O(A \cdot M(s))$. The textbook [BPR06] also covers many aspects discussed in this section.

2.4.1. Basic arithmetic

As a warm up, we consider the complexity of adding, multiplying, and dividing two univariate polynomials. For polynomial arithmetic, we will only use classical arithmetic; faster methods are not considered in this thesis.[13] [vzGG99] contains an extensive treatment of asymptotically optimal algorithms.

In general, there are two natural parameters that affect the complexity of algorithms for polynomials: the (total) degree and the size of the (scalar) coefficients. For a simpler notation, the following definition will be convenient, both for uni- and multivariate polynomials.

Definition 2.4.3 (magnitude of polynomials). We call a univariate or multivariate polynomial f with integer or rational coefficients to be *of magnitude* (n, τ) if its total degree is at most n, and if the maximal bitsize of its scalar coefficients is at most τ. If g is the resulting polynomial of an algorithm whose input polynomials $f_1, \ldots, f_n$ are of magnitude (n, τ), we say that g is of magnitude $O(p(n, \tau), q(n, \tau))$ for some functions $p, q : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$, if $\deg g = O(p(n, \tau))$ and its maximal coefficient bitsize is $O(q(n, \tau))$.

Theorem 2.4.4 (Basic Polynomial Arithmetic). *Let $f, g \in \mathbb{Z}[t]$ of magnitude (n, τ).*
- *Computing $f + g$ requires $n + 1$ additions in $\mathbb{Z}$, the bit complexity is $O(n\tau)$, and $f + g$ is of magnitude $(n, \tau + 1)$*
- *Computing fg requires $O(n^2)$ arithmetic operations in $\mathbb{Z}$, the bit complexity is $O(n^2 M(\tau + \log n))$, and fg is of magnitude $O(n, \tau + \log n)$*

Proof. All results trivially follow by considering Algorithms 2.1 and 2.2. □

Theorem 2.4.5 (Euclidean division for univariate polynomials). *Let $f, g \in \mathbb{Z}[t]$ of magnitude (n, τ). To compute $q, r \in \mathbb{Z}[t]$ such that $f = qg + r$ with $\deg(r) < \deg(g)$ requires $O(n^2)$ arithmetic operations in $\mathbb{Z}$. The bit complexity is $O(n^2 M(n\tau))$, and q and r are of magnitude $O(n, n\tau)$.*

[13] Such fast methods would certainly improve some sub-algorithms, but not the asymptotic bounds of the geometric algorithms that are our focus.

Algorithm 2.2. Univariate multiplication

INPUT: $f, g \in \mathbb{Z}[t]$ of magnitude (n, τ)
OUTPUT: $f \cdot g$

procedure MULT(f, g)
  for $k \in \{0, \dots, 2n\}$ **do**
    $c_k \leftarrow \sum_{j=0}^{n} \operatorname{coef}_{k-j}(f)\operatorname{coef}_j(g)$
  end for
  return $\sum_{k=0}^{2n} c_k t^k$
end procedure

Algorithm 2.3. Univariate division with remainder

INPUT: $f, g \in \mathbb{Z}[t]$ of magnitude (n, τ)
OUTPUT: $q, r \in \mathbb{Q}[t]$ such that $f = qg + r$ and $\deg r < \deg g$

procedure EUCLIDEAN_DIVISION(f, g)
  $p \leftarrow \deg f$, $q \leftarrow \deg g$
  $r \leftarrow f$
  for $k = p, p-1, \dots, q$ **do**
    $c_{k-q} \leftarrow \frac{\operatorname{coef}_j(r)}{b_q}$
    $r \leftarrow r - c_{k-q} t^{k-q} g$ ▷ Remove leading term
  end for
  return $(\sum_{k=0}^{p-q} c_k t^k, r)$
end procedure

Proof. Considering Algorithm 2.3, the number of arithmetic operations easily follows: the iteration is performed at most n times, and one subtraction plus one scalar multiplication is performed in each iteration which has arithmetic complexity $O(n)$. Note that the coefficients of R increase in each iteration by at most τ which proves the statements about bit complexity and magnitude of the output. ☐

The quotient and the remainder of the Euclidean division can be of quite high complexity. This bound can be reduced if g is actually a divisor of f. We use the following measures for polynomials.

Definition 2.4.6 (p-norm). Let $f = \sum_{i=0}^{n} a_i t^i$ with $a_i \in \mathbb{C}$. For $1 \leq p < \infty$, the *p-norm* is defined as

$$\|f\|_p := \sqrt[p]{\sum_{i=0}^{n} |a_i|^p}$$

and the ∞-norm as

$$\|f\|_\infty := \max_{i=0,\dots,n} \{|a_i|\}.$$

Definition 2.4.7 (Mahler measure). Let $f(t) = a_n \prod_{i=0}^{n} (t - \alpha_i)$ be a complex polynomial of degree n. The *Mahler measure* of f is

$$\operatorname{Mea}(f) := |a_n| \prod_{i=1}^{n} \max\{|\alpha_i|, 1\}.$$

The Mahler measure will be of prominent importance in this thesis. We remark that it is multiplicative, which means that

$$\mathrm{Mea}(fg) = \mathrm{Mea}(f) \cdot \mathrm{Mea}(g)$$

for non-constant polynomials.

Lemma 2.4.8. *For any $f \in \mathbb{C}[t]$ of degree n,*

$$\|f\|_\infty \leq \|f\|_1 \leq 2^n \mathrm{Mea}(f) \leq 2^n \|f\|_2.$$

The first inequality is obvious. For a proof of the others, see [BPR06, Prop 10.8 and Prop. 10.9] (The latter is also known as *Landau's inequality*).

Lemma 2.4.9. *Let $f \in \mathbb{Z}[t]$ be a polynomial of magnitude (n, τ) and let $g \in \mathbb{Z}[t]$ be a polynomial that divides f over $\mathbb{Z}[t]$. Then g is of magnitude $O(n, n + \tau)$.*

Proof. Obviously, $\deg g \leq n$. For the bitsize, note that $\mathrm{Mea}(g) \leq \mathrm{Mea}(f)$. It follows that

$$\|g\|_\infty \leq \|g\|_1 \leq 2^n \mathrm{Mea}(g) \leq 2^n \mathrm{Mea}(f) \leq 2^n \sqrt{n+1} 2^\tau.$$

The maximal bitsize of a coefficient of g is $O(\log \|g\|_\infty) = O(n + \tau + \log n) = O(n + \tau)$, which proves the result. □

Another basic algorithm is the evaluation of a polynomial at a given value. Let $q := \frac{c}{d}$ with c, d coprime and $d > 0$. We describe algorithms that compute, for a univariate polynomial $f \in \mathbb{Z}[t]$, $(d^{\deg f} \cdot f(q)) \in \mathbb{Z}$ and for a bivariate polynomial $f \in \mathbb{Z}[x, y]$, $(d^{\deg_y f} \cdot f(x, q)) \in \mathbb{Z}[x]$. Note that in both cases, the initial power of d clears all denominators, hence, the results remain integral objects. This will be convenient for us later, and it does not make a significant difference because we will be only interested in the sign of $f(q)$ (for univariate polynomials) or the roots of the polynomial $f(x, q)$ (for bivariate polynomials). Both values are unaffected by multiplying with a (positive) scalar factor. In Algorithm 2.4, we handle both the uni- and the bivariate cases at once.

Lemma 2.4.10 (polynomial evaluation, univariate case). *Let $f \in \mathbb{Z}[t]$ be of magnitude (n, τ), and $q \in \mathbb{Q}$ with $\mathrm{Bit}(q) = \sigma$. Algorithm 2.4 computes $d^n f(\frac{c}{d})$ and requires $O(n)$ arithmetic operations in $\mathbb{Z}$. The bit complexity is $O(nM(\tau + n\sigma))$ and the result has a bitsize of $O(\tau + n\sigma)$.*

Proof. To show correctness of Algorithm 2.4, note that before the $i+1$-th iteration ($i = 0, \ldots, n$), the variable r in Algorithm 2.4 equals

$$r = a_p c^i + a_{p-1} c^{i-1} d \ldots + a_{p-1} d^i.$$

This proves that the overall result equals $d^n f(\frac{c}{d})$.

The number of arithmetic operations is clearly linear. For the bit complexity, we have to study the bitsize of r in each iteration. Initially, its bitsize is τ. One can easily show by induction that after the i-th iteration, its bitsize is bounded by $i\sigma + \tau + i$, which indicates the bit complexity and the size of the output. □

Algorithm 2.4. Uni- and multivariate rational evaluation

INPUT: $f = \sum_{i=0}^{n} a_i y^i \in D[y]$, with $D = \mathbb{Z}$ or $D = \mathbb{Z}[x]$ of magnitude $(n, \tau), q = \frac{c}{d} \in \mathbb{Q}, d > 0$ with $\mathrm{Bit}(q) = \sigma$
OUTPUT: $d^n f(\frac{c}{d})$

procedure EVAL(f, q)
    $r \leftarrow a_n$, $e \leftarrow 1$
    for $i = 1, \ldots, n$ **do**
        $e \leftarrow de$
        $r \leftarrow c \cdot r + e \cdot a_{n-i}$
    end for
    return r
end procedure

Lemma 2.4.11 (polynomial evaluation, bivariate case). *Let $f \in \mathbb{Z}[x, y]$ be of magnitude (n, τ), and $q \in \mathbb{Q}$ with $\mathrm{Bit}(q) = \sigma$. Algorithm 2.4 computes $d^n f(x, \frac{c}{d})$ and requires $O(n^2)$ arithmetic operations in $\mathbb{Z}$. The bit complexity is $O(n^2 M(\tau + n\sigma))$ and the resulting polynomial is of magnitude $O(n, \tau + n\sigma)$.*

Proof. The correctness follows as for the univariate case and the number of operations over $\mathbb{Z}[x]$ is linear. Note that the only operations necessary in $\mathbb{Z}[x]$ are addition and scalar multiplication, both of which only need $O(n)$ operations. Thus the arithmetic complexity is $O(n^2)$. The bound on the bitsizes follows with the same argument as for the univariate case. □

2.4.2. Computing subresultants

We look next at an efficient method for computing the subresultants of two polynomials f and g, as defined in Section 2.3. The algorithm works by iterated divisions with remainder, and thus resembles the Euclidean algorithm applied on f and g. We concentrate on this technique since it provides the best known complexity bounds. However, we also note an alternative approach that computes all subresultant coefficients by evaluating determinants of suitable matrices. This alternative has been shown to be adequate for polynomial rings with many indeterminates [Ker09a].

We first investigate the size of each subresultant coefficient. For the univariate case, the bound simply follows from applying the Hadamard bound [BPR06, Prop.8.9][Yap00, Lemma 6.27] [vzGG99, Thm.16.6] on the Sylvester submatrices.

Proposition 2.4.12. *For $f, g \in \mathbb{Z}[t]$ of magnitude (n, τ), each coefficient of $\mathrm{Sres}_i(f, g)$ is of size $O(n(\tau + \log n))$.*

For bivariate polynomials, the Hadamard bound can be applied as well. We cite [BPR06, Prop. 8.11 and 8.12] in a simplified form.

Proposition 2.4.13. *Let M be a $d \times d$ matrix where each entry is a univariate polynomial in x of magnitude (n, τ). Then, $\det M$ is a univariate polynomial of magnitude $(nd, (\tau + \log d)d + \log(nd + 1))$.*

Algorithm 2.5. Subresultants by pseudo-remainder

INPUT: $f, g \in D[t]$, $n = \deg f = 1 + \deg g$
OUTPUT: $(\mathrm{Sres}_{n-1}(f,g), \ldots, \mathrm{Sres}_0(f,g))$

procedure SUBRESULTANTS(f, g)
    $d \leftarrow n$
    $S_d \leftarrow f, S_{d-1} \leftarrow g, S_e \leftarrow S_{d-1}$
    while $S_{d-1} \neq 0$ **do**
        $e \leftarrow \deg S_{d-1}, \delta = d - e$
        if $\delta > 1$ **then**
            for $i = 1, \ldots, \delta - 1$ **do** $S_{d-1-i} \leftarrow 0$
            end for
            $S_e \leftarrow \frac{\mathrm{lcf}_{d-1}^{\delta} \mathrm{Sres}_{d-1}}{\mathrm{lcf}_d^{\delta}}$
        end if
        $S_{e-1} \leftarrow \frac{\mathrm{rem}(\mathrm{lcf}_e \cdot \mathrm{lcf}_{d-1} \mathrm{Sres}_d, \mathrm{Sres}_{d-1})}{(-1)^{\delta} \mathrm{lcf}_d^2}$
        $d \leftarrow e$
    end while
    for $i = 1, \ldots, e - 1$ **do** $S_e \leftarrow 0$
    end for
    return$(S_{n-1}, \ldots, S_0)$
end procedure

Applied to the situation of Sylvester matrices, we obtain:

Corollary 2.4.14. *Let $f, g \in \mathbb{Z}[x, y]$ be polynomials of magnitude (n, τ). Then, $\mathrm{Sres}_i(f, g) \in \mathbb{Z}[x][y]$ has a y-degree of at most i, and each coefficient of $\mathrm{Sres}_i(f, g)$ is a univariate polynomial in x of magnitude $O(n^2, n(\tau + \log n))$.*

The structure theorem for subresultants (Theorem 2.3.18) leads to an algorithm to compute the subresultants of f and g as set out in Algorithm 2.5. For simplicity, we restrict its formulation to the case that $\deg f = 1 + \deg g$. See [BPR06, Alg. 8.21] and [Duc00] for complete formulations.

Theorem 2.4.15. *Computing the subresultants of two polynomials $f, g \in D[t]$ with $\deg f = n \geq m = \deg g$ requires $O(nm)$ arithmetic operations in D.*

See [BPR06, Alg. 8.19 and Alg. 8.21] for a proof. We can deduce the bit complexities for univariate polynomials.

Theorem 2.4.16. *Let $f, g \in \mathbb{Z}[t]$ with $\deg f = n \geq m = \deg g$ and all coefficients bounded by 2^τ. The bit complexity for computing the subresultants of f and g is $O(nmM(n(\log n + \tau)))$.*

The bit complexity for bivariate polynomials also follows easily.

Theorem 2.4.17. *Let $f, g \in \mathbb{Z}[x][y]$ both be of magnitude (n, τ). Then, the bit complexity of computing the subresultants of f and g is*

$$O(n^6 M(n(\log n + \tau))).$$

Proof. The number of arithmetic operations in $\mathbb{Z}[x]$ is bounded by $O(n^2)$ according to Theorem 2.4.15. The subresultant polynomials as well as all intermediate results have degree $O(n^2)$, and since each arithmetic operation is at most quadratic in its degree, this gives in total $O(n^6)$ arithmetic operations over $\mathbb{Z}$. The coefficient bitsize of each subresultant is bounded by $O(n(\log n + \sigma))$, and this is also a bound for all intermediate coefficients. The result follows. $\square$

Note that the intermediate results from Algorithm 2.5 are roughly three times as big as the actual subresultant coefficients. Several authors have proposed optimizations in order to reduce this swell-up [Duc00] [LRD00]. The approach presented by Ducos [Duc00] is easy to implement and, according to our experience, the reduced coefficient growth has a significant impact in the practical performance.

2.4.3. Computing the gcd and the square-free part

We have seen in Section 2.3 that $\mathrm{Sres}_k(f,g)$ is a greatest common divisor of f and g for some $k = 0,\ldots,\min\{\deg f, \deg g\}$. We recall that computing "the" gcd of f and g requires a choice of a representative system in order to make the gcd unique. In our applications, however, we are basically interested in the roots of polynomials which obviously do not change when multiplying a constant factor. Thus, what we actually need is the *gcd up to a constant factor* and $\mathrm{Sres}_k(f,g)$ would be a valid choice for that.

Still, the coefficient size of $\mathrm{Sres}_k(f,g)$ is quite bad: By Proposition 2.4.12, it is of magnitude $O(n, n(\log n + \tau))$, but Lemma 2.4.9 shows that there is also a gcd with magnitude $(n, n+\tau)$. One solution would be to make $\mathrm{Sres}_k(f,g)$ primitive, which yields the smallest integer gcd with respect to bitsize, but this causes up to n integer gcd calculations in a post-processing step. With the following lemma [BPR06, Alg.10.1] [GCL92, p.299], we can efficiently find another gcd which has only up to τ more bits than the primitive part of the gcd.

Lemma 2.4.18. *Let $f \in \mathbb{Z}[t]$ be a polynomial of magnitude (n,τ), and $g \in \mathbb{Z}[t]$ such that g divides f over $\mathbb{Q}[t]$, which means that there is a $q \in \mathbb{Q}[t]$ such that $gq = f$. Then, the polynomial $\frac{\mathrm{lcf}(f)g}{\mathrm{lcf}(g)}$ is in $\mathbb{Z}[t]$ and of magnitude $O(n, n+\tau)$.*

Proof. Define the primitive part of g, $\overline{g} := \frac{g}{\mathrm{cont}(g)}$. Obviously, $\overline{g}$ divides f over $\mathbb{Q}[t]$ as well. We claim that $\overline{g}$ also divides f over $\mathbb{Z}[t]$: Assume that $\overline{g}\cdot\overline{q} = f$ for some $\overline{q} \in \mathbb{Q}[t]$. There is some $c \in \mathbb{Q}$ such that $c\overline{q} \in \mathbb{Z}[t]$ and is primitive. Thus, the product $\overline{g}c\overline{q} = cf \in \mathbb{Z}[t]$ is primitive.[14] It follows that $|c| = \frac{1}{\mathrm{cont}(f)}$ (otherwise, cf would not be primitive) and thus, $\overline{q} \in \mathbb{Z}[t]$.

Consequently, $\overline{g}$ divides f over $\mathbb{Z}[t]$ and is thus of magnitude $O(n, n+\tau)$ by Lemma 2.4.9. Furthermore, $\mathrm{lcf}(f) = d \cdot \mathrm{lcf}(\overline{g})$ for some $d \in \mathbb{Z}$, and $\mathrm{Bit}(d) \leq \tau$. It follows that $\frac{\mathrm{lcf}(f)g}{\mathrm{lcf}(g)} = d\overline{g} \in \mathbb{Z}[t]$, therefore, each coefficient of $\frac{\mathrm{lcf}(f)g}{\mathrm{lcf}(g)}$ has at most τ more bits than the corresponding coefficient of $\overline{g}$. The result follows. $\square$

Theorem 2.4.19 (Univariate gcd computation). *Computing a gcd of two polynomials $f, g \in \mathbb{Z}[t]$ of magnitudes (n,τ) and (m,σ) with $n \geq m$ and $\tau \geq \sigma$ requires $O(nm)$*

[14] The fact that the product of primitive polynomials is primitive is known as *Gauss' lemma*. See [Yap00, Lemma 3.1] [vzGG99, Thm.6.6] [BPR06, Lemma 10.16] for proofs.

Algorithm 2.6. gcd of integer polynomials

INPUT: $f_1, f_2 \in \mathbb{Z}[t]$
OUTPUT: $\gcd(f_1, f_2)$ (up to a constant factor)

```
procedure GCD(f_1, f_2)
    Compute the subresultants Sres_0(f_1, f_2), ..., Sres_ℓ(f_1, f_2).    ▷ ℓ := min{deg f_1, deg f − 2}
    k ← min{j ≥ 0 | Sres_j(f_1, f_2) ≠ 0}            ▷ Sres_k(f_1, f_2) is a gcd of f_1 and f_2
    return lcf(f_1)Sres_k(f_1, f_2) / sres_k(f_1, f_2)   ▷ sres_k(f_1, f_2) = lcf(Sres_k(f_1, f_2))
end procedure
```

arithmetic operations in $\mathbb{Z}$. *The bit complexity is* $O(nmM(n(\tau + \log n))$. *The resulting gcd is of magnitude* $O(m, m + \tau)$.

Proof. Consider Algorithm 2.6. The subresultant computation is in the bounds of Theorem 2.4.16. The additional normalization step requires another $2m$ arithmetic operations and the bounds on the bitsize remain the same. The resulting gcd has magnitude $O(m, m + \tau)$ according to Lemma 2.4.18. □

One could replace $\mathrm{lcf}(f)$ by $\gcd(\mathrm{lcf}(f), \mathrm{lcf}(g))$ in the last step of the algorithm to obtain an even smaller return value.

We next consider the square-free part of a univariate polynomial. The square-free part $\overline{f}$ of f is a square-free polynomial such that $V_{\mathbb{C}}(\overline{f}) = V_{\mathbb{C}}(f)$. In other words, if $f = \mathrm{lcf}(f) \prod_{i=0}^{r} (X - \alpha_i)^{e_i}$, the square-free part is, up to a constant factor, defined as $\overline{f} = \prod_{i=0}^{r} (X - \alpha_i)$. The non-uniqueness of the square-free part does not pose any problems for our applications. The square-free part of a polynomial will be of importance in Section 2.4.4, where we discuss an efficient method to compute the real roots of a polynomial.

Lemma 2.4.20. *Let* $f \in \mathbb{Z}[t]$ *be of magnitude* (n, τ), *and* $g := \gcd(f, f')$, *as returned by Algorithm 2.6. Then,* $\frac{\mathrm{lcf}(f) f}{g} \in \mathbb{Z}[t]$ *is the square-free part of* f, *and is of magnitude* $O(n, n + \tau)$.

Proof. Let $f = \mathrm{lcf}(f) \prod_{i=0}^{r} (X - \alpha_i)^{e_i}$, and let g be the gcd of f and f' returned by Algorithm 2.6. We have $g = \mathrm{lcf}(g) \prod_{i=0}^{r} (X - \alpha_i)^{e_i - 1}$, thus $\frac{f}{g}$ is the square-free part of f. Reconsider the proof of Lemma 2.4.18: We have shown that $g = d\overline{g}$, where $\overline{g}$ is the primitive part of g, and d is an integer that divides $\mathrm{lcf}(f)$. Thus, since $\overline{g}$ divides f over $\mathbb{Z}[t]$, we have

$$\frac{\mathrm{lcf}(f) f}{g} = \underbrace{\frac{\mathrm{lcf}(f)}{d}}_{\in \mathbb{Z}} \underbrace{\frac{f}{\overline{g}}}_{\in \mathbb{Z}[t]} .$$

The bitsize of the first factor is bounded by τ, and the magnitude of the second factor is $O(n, n + \tau)$ from Lemma 2.4.9. □

Theorem 2.4.21 (univariate square-free part computation). *Computing the square-free part of a polynomial* $f \in \mathbb{Z}[t]$ *of magnitude* (n, τ) *requires* $O(n^2)$ *arithmetic operations in* $\mathbb{Z}$. *The bit complexity is* $O(n^2 M(\tau + \log n))$. *The resulting square-free part is of magnitude* $O(n, n + \tau)$.

Algorithm 2.7. Square-free part of integer polynomials

INPUT: $f \in \mathbb{Z}[t]$

OUTPUT: Square-free part of f (up to a constant factor)

```
procedure SQUARE_FREE_PART(f)
    g ← gcd(f, f')                                   ▷ Algorithm 2.6
    return lcf(f)f / g
end procedure
```

Proof. Consider Algorithm 2.7. Computing the derivative f' requires $n-1$ multiplications in $\mathbb{Z}$, and f' is of magnitude $(n, \tau + \log n)$. Thus, Algorithm 2.6 is applied on polynomials of magnitude $(n, \tau + \log n)$, which yields $O(n^2)$ arithmetic operations and a bit complexity of $O(n^2 M(\tau + \log n))$. The magnitude of the result follows from Lemma 2.4.20. □

An alternative way to compute the square-free part is to consider the cofactors arising in the subresultant sequence (Definition 2.3.15).

Lemma 2.4.22. *For $f \in D[t]$, let $k = \deg\gcd(f, f') > 1$. Then, $v_{k-1}(f, f')$ is the square-free part of f.*

Proof. Note that f has precisely $n-k$ distinct complex roots. Since $\mathrm{Sres}_{k-1}(f, f') = 0$ by the structure theorem (Theorem 2.3.18), it holds that

$$u_{k-1}(f, f') \cdot f + v_{k-1}(f, f') \cdot f' = 0$$

and $\deg v_{k-1}(f, g) \leq n - k$. It follows that f divides the product $v_{k-1}(f, f') \cdot f'$ and $v_{k-1}(f, f')$ must contain each root of f at least once, which proves the theorem. □

A related object is the *square-free factorization* of f.

Definition 2.4.23 (square-free factorization). A polynomial $f \in D[t]$ can be decomposed into

$$f = \prod_{i=0}^{m} f_i^i$$

where the f_i's are square-free and pairwise coprime. This decomposition is called *square-free factorization.* Note that f_i contains precisely the roots of f with multiplicity i.

Theorem 2.4.24. *Algorithm 2.8 (see also [Yun76] [GCL92, Alg.8.2] [vzGG99, Alg. 14.21]) computes a square-free factorization of $f \in \mathbb{Z}[t]$ with $O(n^2)$ operations in $\mathbb{Z}$.*

See [vzGG99, Thm.14.23] for a proof. The correctness statement is also proved in [GCL92, p.341]. The overhead of computing a square-free factorization does not change the (arithmetic) complexity, compared to just computing the square-free part.

It is in practice advantageous to work with the square-free factorization instead of using the square-free part, because the polynomial is split into smaller parts, which can be processed faster in subsequent operations. However, it is mostly more convenient for theoretical considerations to work with a single object, so we will use the square-free part for the complexity analysis.

Algorithm 2.8. Yun's square-free factorization

INPUT: $f \in \mathbb{Z}[t]$ primitive
OUTPUT: Polynomials $f_1, \ldots, f_m$ (for some $m > 0$) such that $f = \prod f_i^i$ (up to constant factor)

procedure SQUARE_FREE_FACTORIZATION(f)
  $i \leftarrow 1$
  $c \leftarrow \gcd(f, f')$
  $w \leftarrow \frac{f}{c}$
  $y \leftarrow \frac{f'}{c}$
  $z \leftarrow y - w'$
  while $z \neq 0$ **do**
    $f_i \leftarrow \gcd(w, z)$
    $i \leftarrow i + 1$
    $w \leftarrow \frac{w}{g}$
    $y \leftarrow \frac{z}{g}$
    $z \leftarrow y - w'$
  end while
  return$(f_1, \ldots, f_{i-1}, w)$
end procedure

2.4.4. Root isolation

We turn to the problem of isolating the real roots of a real univariate polynomial.

Definition 2.4.25 (isolating interval). Let $f \in \mathbb{R}[x]$ and $\alpha \in \mathbb{R}$ be a root of f. An interval I is *isolating* for f and α if $I \cap V(f) = \{\alpha\}$.

The *(real) root isolation problem* to find disjoint isolating intervals for all real roots of a real polynomial $f \in \mathbb{R}[t]$. The general approach for this problem is as follows. Start with an interval that is sufficiently large to contain all real roots of f. Then, start to subdivide the interval into subintervals. Throw away intervals that provably do not contain a root. Report intervals that provably contain exactly one root. Keep on subdividing intervals that might contain more than one root.

Three classical approaches that use this scheme can be identified. The first one is based on Sturm's theorem (Theorem 2.3.25) to count the number of roots inside an interval. The other two methods are both based on *Descartes' rule of signs* (Theorem 2.4.26), but the two methods differ in the way of subdividing intervals: the *Descartes method* always subdivides at the midpoint, whereas the *Continued Fraction* approach uses a non-uniform scheme based on the lower bounds for the positive real roots of polynomials. An experimental comparison of various root solving techniques has been appeared recently [EHK$^+$09].

We will focus on the Descartes method, whose modern formulation goes back to Collins and Akritas [CA76].[15] This introductory section clearly cannot cover all aspects of the

[15] We remark that the term "Descartes method" is not undisputed in the community. Akritas [Akr] advocates the name "Vincent-Collins-Akritas (VCA) bisection method", because he considers the contribution of Vincent [Vin36] to be crucial for the method. We decided to use the term "Descartes method" anyway, since Descartes' rule of signs is undisputably the main tool needed for the algorithm. Note that we do not use the term "Descartes' method" (with apostrophe) since this would wrongly indicate the algorithm was

topic; see Eigenwillig's thesis [Eig08] for an excellent treatment of the Descartes method, with complete historic references and discussions with alternative approaches.

The Descartes method is based on an upper bound for the positive real roots of a polynomial.

Theorem 2.4.26 (Descartes' rule of signs). *Let $f = \sum_{i=0}^{n} a_i t^i \in \mathbb{R}[t]$ be a polynomial of degree n. Then $\mathrm{Var}(a_0, \ldots, a_n)$, the number of sign variations (see Definition 2.3.24), exceeds the number of positive real roots of f, counted with multiplicities, by a non-negative even number.*

In particular, if $\mathrm{Var}(a_0, \ldots, a_n) = 0$, f has no positive real root, and if $\mathrm{Var}(a_0, \ldots, a_n) = 1$, f has exactly one positive real root (which is of multiplicity one).

See, for instance, [Eig08, Theorem 2.2], [BPR06, Theorem 2.33], [CA76] for proofs. In this form, the rule is only applicable for counting the roots inside the interval $(0, \infty)$. However, it can be simply extended to count the roots of any open interval (c, d).

Proposition 2.4.27. *For $c, d \in \mathbb{R}$, $c < d$, consider the Möbius transformation*

$$\varphi_{c,d} : (0, \infty) \to (c, d), t \mapsto \frac{ct + d}{t + 1}$$

and the polynomial

$$\mathcal{T}_{f,c,d}(t) := (t + 1)^{\deg(f)} \cdot (f \circ \varphi_{c,d})(t).$$

$\mathrm{Var}(\mathcal{T}_{f,c,d})$ exceeds the number of real roots of f inside (c, d), counted with multiplicities, by a non-negative even number.

Proof. $\varphi_{c,d}$ is a bijection between the interval (c, d) and $(0, \infty)$, thus, the roots of f inside (c, d) are in one-to-one correspondence to the positive real roots of $\mathcal{T}_{f,c,d}(t)$ (the initial term $(t + 1)^{\deg f}$ is necessary to clear denominators). The result follows by Descartes' rule of signs. □

For an interval $I = (c, d)$, we write $\mathrm{Var}(f, I)$ instead of $\mathrm{Var}(\mathcal{T}_{f,c,d})$ for simplicity. The Descartes method to isolate the roots inside some interval I is straightforward to formulate, see Algorithm 2.9. Several things are noteworthy about this algorithm.

- The correctness of the Descartes method follows from Descartes' rule of signs. That is, the output list indeed only contains isolating intervals, and if the algorithm terminates, it must necessarily return all real roots. However, we have no guarantee so far that indeed each root is eventually encapsulated in an interval J such that $\mathrm{Var}(f, J) = 1$. On the contrary, if f has multiple roots inside I, the algorithm always diverges, since Descartes' rule of signs counts with multiplicities. We will show below that the algorithm always terminates if all real roots of f inside I are simple.
- Typically, one is interested in all real roots of f. Thus, one needs to pass an initial interval containing all real roots. This can be chosen using, for instance, Theorem 2.2.11 . However, there exist much better root bounds for the practice. We recommend to replace the bound of Theorem 2.2.11 by the *Fujiwara bound*

$$2 \cdot \max\left\{\left|\frac{a_{n-1}}{a_n}\right|, \sqrt{\left|\frac{a_{n-2}}{a_n}\right|}, \ldots, \sqrt[n-1]{\left|\frac{a_1}{a_n}\right|}, \sqrt[n]{\left|\frac{a_0}{2a_n}\right|}\right\}$$

already foreseen by Descartes.

Algorithm 2.9. The Descartes method

INPUT: $f \in \mathbb{R}[t]$, $I \subset \mathbb{R}$
OUTPUT: Isolating intervals for all roots of f inside I

procedure DESCARTES(f, I)
  $Q \leftarrow \{I\}$ ▷ The queue of active intervals
  while Q is not empty **do**
    Let $J = (c, d)$ be the first element of Q. Remove J from Q.
    $v \leftarrow \mathrm{Var}(f, J)$
    if $v = 0$, do nothing
    if $v = 1$, append I to the output list
    if v>1 **then**
      $m \leftarrow \frac{c+d}{2}$
      if $f(m) = 0$, append $[m, m]$ to the output list
      Append (c, m) and (m, d) to Q
    end if
  end while
end procedure

in a concrete implementation. A more in-depth discussion on root bounds, including a proof that the Fujiwara bound is indeed a root bound, can be found in [Eig08, §2.4].

- Each subdivision step halves the size of an interval, and the bitsize of the boundary point of the interval also increases by one. Therefore, if the initial interval of the form $(-2^\tau, 2^\tau)$ is chosen, one needs at most $O(\tau + \ell)$ bits to represent an interval of size $2^{-\ell}$. Since there are $2^{\tau+\ell}$ intervals of this size within $(-2^\tau, 2^\tau)$, this bound is optimal.
- Different from other descriptions of the Descartes method, Algorithm 2.9 is designed to always subdivide one of the largest active subintervals. This means that the *subdivision tree* induced by the algorithm is traversed in a BFS-manner. This traversal strategy looks arbitrary here but becomes crucial for the variants of the Descartes method that cope with multiple roots in Section 2.6.2
- In an efficient implementation of the algorithm, carrying out the transformation $\mathcal{T}_{f,c,d}$ (needed to compute $\mathrm{Var}(f, J)$) from scratch in each iteration must be avoided. Instead, with each interval $J = (c, d)$ in the queue, the algorithm also stores a transformed polynomial $f_J(t) := f(c + t(d - c))$ such that the roots of f_J in the unit interval correspond to the roots of f in J. In this way, $\mathrm{Var}(f, J)$ can be computed for any subinterval by the transformation $t^n f_J(\frac{1}{t+1})$. The costs for this step are dominated by the *Taylor shift*, that is, the expansion of $g(x + 1)$ for a polynomial g. Moreover, if J is subdivided into J_1 and J_2, the polynomials J_1 and J_2 are easily computatble from f_J by
$$f_{J_1} = 2^n f_J(t/2), \quad f_{J_2} = f_{J_1}(x + 1).$$
Again, the costs are dominated by a Taylor shift. We refer the reader to [Eig08, §3.2.4] for details.

Let us assume that f only has simple roots within the input interval I. The standard proof for the termination of Algorithm 2.9 is based on the two theorems, called the *one-circle theorem* and the *two-circle theorem* [Eig08, Prop. 2.33 and Prop. 2.34] [KM06] [ESY08]. In [BPR06, Theorem 10.44], both results are combined into a "three-circle theorem".

Theorem 2.4.28. *Let f be as above, and $J = (c, d)$.*

- ***One-circle Theorem:*** *Let C be the (unique) open complex disc with diameter $d - c$. If f has no root in C, then $\mathrm{Var}(f, J) = 0$ (See the left of Figure 2.8).*
- ***Two-circle Theorem:*** *Let C_1 and C_2 be the open circles that circumscribe the two equilateral triangles having cd as one side. If f has exactly one (simple) root in the union $C_1 \cup C_2$, then $\mathrm{Var}(f, J) = 1$ (See the right of Figure 2.8).*

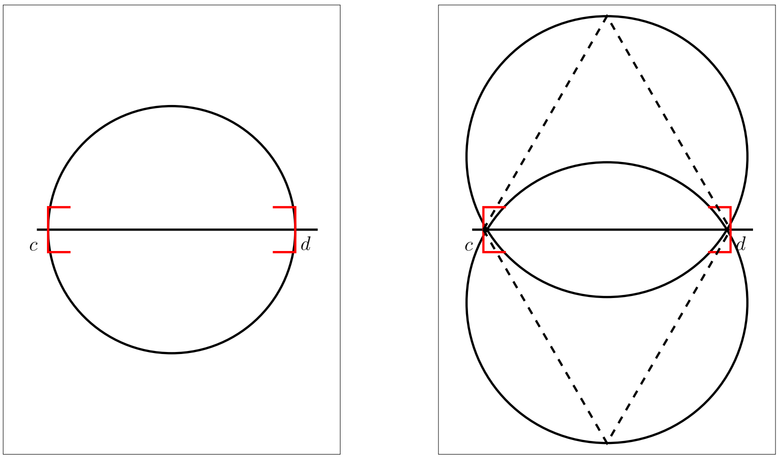


Figure 2.8. On the left: If the circle does not contain a root of f, then $\mathrm{Var}(f, I) = 0$. On the right: If the union of the circles contains exactly one root, then $\mathrm{Var}(f, I) = 1$.

Both the one-circle and the two-circle theorems are subsumed in a more general result that goes back to Obreshkoff [Obr03].

Definition 2.4.29 (Obreshkoff disc, Obreshkoff lens, Obreshkoff area). Let $J = (c, d)$ be an interval, and let ℓ denote the real line in the complex plane. For $q = 0, \ldots, n$, the Obreshkoff disc $\overline{C_q} \subset \mathbb{C}$ is the unique disc with c and d at its boundary that makes an angle of $\alpha := \frac{\pi}{q+2}$ at d with ℓ and whose center is above ℓ. Likewise, $\underline{C_q} \subset \mathbb{C}$ is the Obreshkoff disc with the same properties but with its center below ℓ. The *Obreshkoff lens* L_q is the interior of $\overline{C_q} \cap \underline{C_q}$ and the Obreshkoff area A_q is the interior of $\overline{C_q} \cup \underline{C_q}$ (Figure 2.9).

First of all, observe that $L_n \subset \ldots \subset L_0 \subset A_0 \subset \ldots \subset A_n$, so any Obreshkoff lens is contained in any Obreshkoff area. Moreover, it is well known that for any point p on the boundary of $\overline{C_q}$, except c or d, the angle $\angle cpd$ is the same (this is usually called the *inscribed angle theorem*), and that this angle equals $\alpha = \frac{\pi}{q+2}$. It follows that for $q = 0$, the Obreshkoff discs $\overline{C_0}$ and $\underline{C_0}$ coincide, and the Obreshkoff area A_0 is precisely the disc in the one-circle theorem. Moreover, for $q = 1$, it follows that $\alpha = \frac{1}{3}\pi$ and thus, A_1 coincides with the region defined in the two-circle theorem.

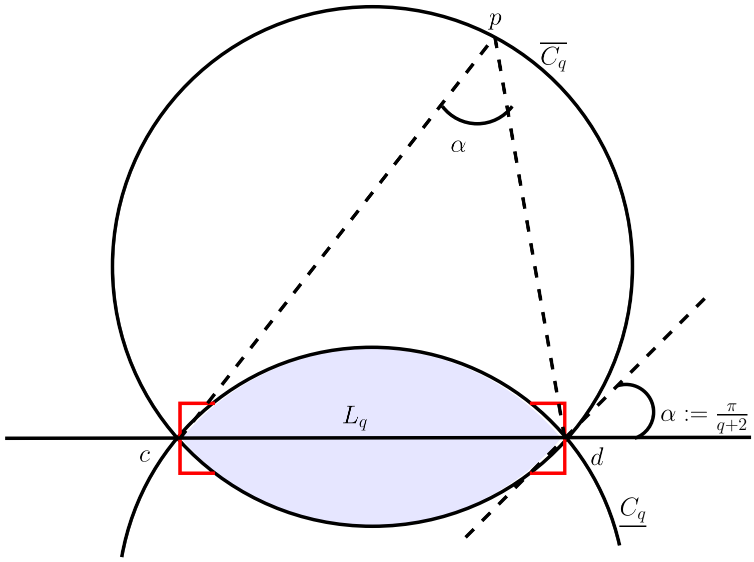


Figure 2.9. The Obreshkoff disc $\overline{C_q}$ and the Obreshkoff lens L_q

Theorem 2.4.30 (Obreshkoff's theorem). *Let $f \in \mathbb{R}[t]$ be of degree n, and $J = (c, d)$ be an interval with $\operatorname{Var}(f, J) = v$. If the Obreshkoff lens L_{n-q} contains at least q roots of f, counted with multiplicity, then $v \geq q$. If the Obreshkoff area A_q contains at most q roots of f, counted with multiplicities, then $v \leq q$. In particular, if J contains exactly q roots in L_{n-q}, and no further root in $A_q \setminus L_{n-q}$, then $v = q$.*

As a consequence, we can state more informally that $\operatorname{Var}(f, J)$ counts at least all roots that are contained in L_n, and it counts at most the roots that are contained in A_n. Theorem 2.4.30 implies that if f has a multiple root of multiplicity q, the Descartes method eventually counts exactly q for the interval containing the root (after sufficiently many subdivisions). Although this does not avoid divergence in Algorithm 2.9, it is of great importance later for our variants that cope with multiple roots.

We turn to the complexity analysis of the Descartes method. We do not give full details for the analysis, but several results along the way will be useful in different contexts, thus we sketch the main steps of the analysis. It is helpful to consider the subdivision process as a tree that we refer to as the *subdivision tree* . The root of the tree is the input interval I, the children of a node are those intervals that are appended to the queue. Thus, leaves of the tree are those intervals J such that $\operatorname{Var}(f, J) \in \{0, 1\}$.

A first important observation is that the subdivision tree has a width of at most $n = \deg f$ on each level, by the following theorem [Sch34].

Theorem 2.4.31. *Let J be an interval and $J_1, \ldots, J_\ell$ a partition of J into ℓ subintervals. Then,*

$$\operatorname{Var}(f, J) \geq \sum_{i=0}^{\ell} \operatorname{Var}(f, J_i).$$

An elementary way of proving this is to consider f with respect to the *Bernstein basis*, and exploiting the properties of the *De Casteljau* algorithm. Since we do not discuss these concepts in detail, see [Eig08, Corollary 2.27] for a proof.

The crucial quantity that controls the depth of the subdivision tree is the *separation* of the polynomial.

Definition 2.4.32 (separation). For $f \in \mathbb{R}[t]$ with at least two distinct roots, the *separation* is defined as the minimal distance between any two distinct complex roots of f, that is,

$$\operatorname{sep}(f) := \min\{|\alpha_i - \alpha_j| \mid \alpha_i, \alpha_j \in V_{\mathbb{C}}(f), \alpha_i \neq \alpha_j\}.$$

For convenience, we define $L_f := \log \frac{1}{\operatorname{sep}(f)}$.

Once the discs in the one-circle and two-circle theorems have a diameter smaller than $\operatorname{sep}(f)/4$, the subdivision stops. Consequently, the tree depth is bounded by $O(\tau + L_f)$.

The following result allows us to bound the magnitude of the separation of a polynomial. In this general form, it first appeared in [Eig08, Thm. 3.9].[16]

Theorem 2.4.33 (generalized Davenport-Mahler bound). *Let $f \in \mathbb{C}[t]$ with $n := \deg f \geq 2$ and exactly $r \leq n$ distinct complex roots $V := \{\alpha_1, \ldots, \alpha_r\}$. Let $G = (V, E)$ be a directed graph on the roots such that:*

- *for every edge $(\alpha, \beta) \in E$, it holds $|\alpha| \leq |\beta|$*
- *G is acyclic, and*
- *the in-degree of any node is at most 1.*

In this situation

$$\prod_{(\alpha,\beta)\in E} |\alpha - \beta| \geq \frac{\sqrt{|\operatorname{sres}_{n-r}(f, f')|}}{\sqrt{|\operatorname{lcf}(f)|\operatorname{Mea}(f)^{r-1}}} \cdot \left(\frac{\sqrt{3}}{r}\right)^{\#E} \cdot \left(\frac{1}{r}\right)^{r/2} \cdot \left(\frac{1}{\sqrt{3}}\right)^{\min\{n, 2n-2r\}/3}.$$

As a corollary, we can bound L_f and thus, the depth of the subdivision tree.

Corollary 2.4.34. *For $f \in \mathbb{Z}[t]$, which is not necessarily square-free, we have*

$$L_f = \log \frac{1}{\operatorname{sep}(f)} = O(n(\tau + \log n)).$$

Proof. Consider a pair of roots (α, β) of f such that their distance equals the separation of f. After a possible swap, the conditions of Theorem 2.4.33 are satisfied, and consequently,

$$\log \frac{1}{|\alpha - \beta|} \leq \log \frac{\sqrt{\operatorname{lcf}(f)}\operatorname{Mea}(f)^{r-1}}{\sqrt{|\operatorname{sres}_{n-r}(f)|}} + O(\log r) + O(r \log r) + O(n).$$

Since f has precisely r distinct complex roots, $\gcd(f, f') = m - r$, thus, $\operatorname{sres}_{n-r}(f, f') \neq 0$. Moreover, $\operatorname{sres}_{n-r}(f, f') \in \mathbb{Z}$, since f is an integer polynomial. Therefore, we can bound $\operatorname{sres}_{n-r}(f, f') \geq 1$. It follows that

$$\log \frac{1}{|\alpha - \beta|} = O(\tau + n \log \operatorname{Mea}(f) + n \log n).$$

By Lemma 2.4.8, it also follows that

$$\log \operatorname{Mea}(f) \leq \log \|f\|_2 \leq \log \sqrt{n}\|f\|_\infty = O(\tau + \log n). \quad \square$$

[16] Note that the description given in [Eig08] uses *subdiscriminants*, which are not discussed in this thesis. However, we can use the identity $|\operatorname{lcf}(f) \cdot \operatorname{sdisc}_i(f, g)| = |\operatorname{sres}_i(f, g)|$ as proven in [BPR06, Prop.4.27] to restate the formula in terms of subresultants.

Still, it does not give optimal complexity to bound the tree size by multiplying maximal depth with maximal width. By exploiting the Davenport-Mahler bound more carefully, an improved estimation is possible.

Theorem 2.4.35 (Eigenwillig, Sharma, and Yap 2006). *For a polynomial $f \in \mathbb{Z}[t]$ with degree n and maximal bitsize τ, the size of the subdivision tree is bounded by $O(n(\tau + \log n))$.*

This shows that the subdivision tree is extremely sparse, since we have the same (asymptotic) bound for its depth as for its total size.

The idea of the proof is as follows: a node in the subdivision tree is called *terminal* if both its children are leaves. For each terminal node, one can assign a unique pair of roots of f that is responsible for the subdivision of the node. The depth of the terminal node in the tree is determined by the distance of the roots. Finally, one can arrange the roots in a way that the conditions of Theorem 2.4.33 are satisfied and bound the product of the root distances (corresponding to the sum of the depth of the terminal nodes) with the same complexity bounds as for the separation. See [ESY08] or [Eig08, §3.1.5 and §3.2.2] for the complete description. For the final runtime analysis, the following operation is crucial.

Definition 2.4.36 (Taylor shift). For $f \in \mathbb{Z}[t]$, the *Taylor shift* is the operation that computes the coefficients of $f(t+1) \in \mathbb{Z}[t]$. We denote by $T(n, \tau)$ the bit complexity to perform a Taylor shift for a polynomial of magnitude (n, τ).

It is well known that $T(n, \tau) = O(n^2(\tau + n))$ (e.g., [Joh91]). Such methods only perform additions over $\mathbb{Z}$ and are usually called *classical Taylor shifts*. More sophisticated approaches achieve a bound of $T(n, \tau) = O(M(n(n + \tau) \log n))$ [vzGG97]. Such methods are called *(asymptotically) fast Taylor shifts*.

Theorem 2.4.37. *For a square-free polynomial f of magnitude (n, τ), and an initial interval whose boundaries are of bitsize $O(\tau)$, the Descartes method isolates the real roots with bit complexity $O(n(\tau + \log n)T(n, n^2(\tau + \log n)))$.*

This bound can be achieved in the following way: The costs per node are dominated by the computation of the Möbius transformation $\mathcal{T}_{f,c,d}$. In turn, this cost is determined by the cost of a *Taylor shift* (the other operations are easily shown to be of linear complexity). With the bound on the tree size from Theorem 2.4.35, this yields $O(T(n, \sigma)n(\tau + \log n))$ bit operations in $\mathbb{Z}$, where σ is an upper bound on the bitsizes during the algorithm.

The bitsize within a Taylor shift is determined by τ in the first step and grows by n per subdivision level. Because the depth of the tree is bounded by $O(n(\tau + \log n))$, the maximal bitsize is bounded by $O(n^2(\tau + \log n))$. The analysis is worked out in detail in [Eig08, §3.2.4].

For not-necessarily square-free polynomials, we can now prove the following theorem.

Theorem 2.4.38. *Let $f \in \mathbb{Z}[t]$ be of magnitude (n, τ). Isolating all real roots of f can be done with bit complexity $O(n(n + \tau)T(n, n^2(n + \tau)))$.*

Proof. Consider Algorithm 2.10. Computing the square-free part has a bit complexity of $O(n^3 M(\tau + \log n)) = O(n^3(\tau + \log n)^2)$, according to Theorem 2.4.21, and $\overline{f}$ is of magnitude $O(n, n + \tau)$. Applying the Descartes method for this polynomial has a bit complexity of

Algorithm 2.10. Real root isolation

INPUT: $f = \sum_{i=0}^{n} a_i t^i \in \mathbb{Z}[t]$
OUTPUT: Isolating intervals for all real roots of f

procedure SOLVE(f)
  $\overline{f} \leftarrow$ SQUARE_FREE_PART(f)
  $b \leftarrow \lceil \log |1 + \max_{i=0,\dots,n-1}\{\frac{a_i}{a_n}\}| \rceil$
  $I \leftarrow (-2^b, 0)$ ▷ I contains all negative roots of f
  $I_1^-, \dots, I_r^- \leftarrow$ DESCARTES($\overline{f}, I$)
  $I \leftarrow (0, 2^b)$ ▷ I contains all positive roots of f
  $I_1^+, \dots, I_s^+ \leftarrow$ DESCARTES($\overline{f}, I$)
  if f(0)=0 **then return** $I_1^-, \dots, I_r^-, [0,0], I_1^+, \dots, I_s^+$
  elsereturn $I_1^-, \dots, I_r^-, I_1^+, \dots, I_s^+$
  end if
end procedure

$O(n(n+\tau)T(n, n^2(n+\tau)))$. □

We now analyze another quantity, namely the bitsize of the returned intervals.

Definition 2.4.39 (standard interval). We call an interval I a *standard interval* if there exists an integer $\ell \geq 0$ such that I can be written as $I = (\frac{a}{2^\ell}, \frac{a+1}{2^\ell})$ with $\ell \in \mathbb{Z}$. The bitsize of a standard interval is given by $\mathrm{Bit}(I) := \ell + \mathrm{Bit}(a)$.

Proposition 2.4.40. *Let $I = (c, d)$ be a standard interval with bitsize σ, and let $m := \frac{c+d}{2}$ be the midpoint of I. Then, both (c, m) and (m, d) are standard intervals with bitsize $\sigma + 2$.*

It follows that all intervals produced by Algorithm 2.10 are standard intervals, since the initial intervals are chosen to be standard intervals (this is the reason why Algorithm 2.10 performs separate Descartes instances on the negative and the positive numbers).

Lemma 2.4.41. *Let I be an isolating interval for f returned by Algorithm 2.10. Then* $\mathrm{Bit}(I) = O(\tau + L_f) = O(n(\tau + \log n))$.

Proof. The bitsize of the initial interval is bounded by τ, and the interval passes through up to L_f bisections until it is returned. □

Summary

This section has dealt with fundamental algorithms for integer polynomials. Most importantly, we have learned about algorithms for computing the subresultant of polynomials, the greatest common divisor, the square-free part, and last but not least, for isolating the real roots of a polynomial.

2.5. Computation with algebraic numbers

The Descartes method, explained in Section 2.4.4, allows us to compute isolating intervals for the real roots of polynomials. Note that, in turn, any algebraic number α is uniquely defined by a pair (f, I) where the *defining polynomial* f of α satisfies $f(\alpha) = 0$ and where I is an isolating interval for α and f. Throughout this thesis, we will represent algebraic numbers by such pairs, and call (f, I) the *isolating interval representation* of α. This representation of algebraic numbers appears in the literature for a long time [Loo82a]; the perhaps more intuitive way of representing algebraic numbers by nested root-expressions is known to be incomplete, meaning that such expressions cannot cover all real algebraic numbers.

We emphasize that we do not require the defining polynomial f for α to be the *minimal polynomial* m_α of α, that is, we do not require irreducibility. Of course, f must be a multiple of m_α, and in a practical implementation, whenever f is known to factorize into two factors f_1 and f_2, one should check whether α is a root of f_1 or f_2, and change its representation with respect to the obtained smaller factor.

For all our algorithms, we require the isolating interval to be a standard interval. Note that Algorithm 2.10 returns standard intervals. We will explore several methods to compute with algebraic numbers in isolating interval representation: We show how the isolating interval can be refined to arbitrarily small size (which is equivalent to numerically approximating an algebraic number). We also discuss how to compare algebraic numbers in this representation. Finally, we examine how to approximate $g(\alpha)$ (g being some univariate polynomial) to any precision, and in particular, how to evaluate the sign of this expression. For this last operation, we will need some basic properties of *interval arithmetic*.

2.5.1. Root refinement

We first look at the problem of refining an isolating interval. That is, if I is an isolating interval for a root α of a square-free f, we want to compute an interval $J \subsetneq I$ that contains α. Also, the refinement method shall come with the property that $\mathrm{width}(J) \leq c \cdot \mathrm{width}(I)$ with a constant $c < 1$, so that an iterated refinement process yields an isolating interval of arbitrarily small size. In the following, we denote the width of an interval by $\mathrm{w}(I)$.

We consider two solutions to this problem. First, we talk about the well-known *bisection*, which halves the interval in every step. As a more sophisticated solution, we consider the *quadratic interval refinement* method which is a hybrid of bisection and the secant method.

Bisection. This is the simplest method for refining an isolating interval. Let (c, d) be isolating for α, then $f(c)f(d) < 0$ since f is square-free. We compute $f(m)$ with $m := \frac{c+d}{2}$. If $f(m) = 0$, the root is computed explicitly (the best approximation one can get). Otherwise, it is checked whether $f(m)f(c) < 0$. If so, (c, m) is chosen as the new isolating interval, otherwise (m, d) is chosen. Clearly, this method halves the isolating interval in each iteration. The complexity of one bisection step is determined by the evaluation of f at m.

Proposition 2.5.1. *Let $f \in \mathbb{Z}[t]$ be of magnitude (n, τ), and let I be an isolating standard interval of bitsize σ. Using Algorithm 2.11, one bisection step requires $O(n)$ arithmetic*

Algorithm 2.11. Bisection method

INPUT: $f \in \mathbb{Z}[t]$ square-free, $I = (c, d)$ isolating
OUTPUT: $J \subset I$ isolating

procedure BISECTION$(f, (c, d))$
  $m \leftarrow \frac{c+d}{2}$
  $s \leftarrow \mathrm{sign}(f(m))$
  if $s = 0$, **return** $[m, m]$
  if $s = \mathrm{sign}(f(c))$, **return** (m, d) **else return** (c, m)
end procedure

Algorithm 2.12. Quadratic interval refinement

INPUT: $f \in \mathbb{Z}[t]$ square-free, $I = (c, d)$ isolating, $N = 2^{2^i} \in \mathbb{N}$
OUTPUT: (J, N') with J isolating and $N' \in \mathbb{N}$

procedure QIR$(f, I = (c, d), N)$
  if $N = 2$, **return** (BISECTION(f, I),4).
  $w \leftarrow \frac{d-c}{N}$
  $m' \leftarrow c + \mathrm{round}(N \frac{f(c)}{f(c)-f(d)})w$ $\triangleright\ m = c + \frac{f(c)}{f(c)-f(d)}(c - d)$
  $s \leftarrow \mathrm{sign}(f(m'))$
  if $s = 0$, **return** $([m', m'], \infty)$
  if$s = \mathrm{sign}(f(c))$ **and** $\mathrm{sign}(f(m' + w)) = \mathrm{sign}(f(d))$, **return** $([m', m' + w], N^2)$
  if$s = \mathrm{sign}(f(d))$ **and** $\mathrm{sign}(f(m' - w)) = \mathrm{sign}(f(c))$, **return** $([m' - w, m'], N^2)$
  Otherwise, return $(I, \sqrt{N})$.
end procedure

operations in $\mathbb{Z}$. The bit complexity is $O(nM(\tau + n\sigma))$, and the resulting interval is a standard interval of bitsize $\sigma + 2$.

Proof. The complexity bounds follow from Lemma 2.4.10 (univariate polynomial evaluation), and the resulting interval is a standard interval of bitsize $\sigma + 2$ by the same argument as in Proposition 2.4.40. □

Quadratic interval refinement (qir). Abbott [Abb06] has presented a method which can be considered as a hybrid of bisection and the secant method, with eventual quadratic convergence: Consider an isolating interval (c, d) for α and f of width δ. The idea is as follows: the interval (c, d) is (conceptually) divided into N subintervals of size $\frac{\delta}{N}$, for some $N \in \mathbb{N}$. The secant through the points $(c, f(c))$ and $(d, f(d))$ intersects the x-axis once between c and d since $f(c)f(d) < 0$. Let m be the x-value of that intersection. It is expected that m is close to α. Therefore, we look for the closest subdivision point m' to m and check whether one of the intervals incident to m' has opposite signs at the boundaries. If yes, we take the corresponding interval as the new isolating interval, and square N for the next iteration. If no sign change was detected, N is set to $\sqrt{N}$ for the next iteration, and the isolating interval remains unchanged.

In our formulation of the algorithm, a call of the qir algorithm does not necessarily refine an isolating interval. However, in this case, N is decreased as a side effect – considering a sequence of qir calls, the method will refine the interval when N decreases to 2 at the latest (because a bisection step is performed in this case). Note that we implicitly assume that the returned value of N is always passed in the next call of qir, and that a qir call with $N > 4$ only takes place if a preceding qir call for $\sqrt{N}$ succeeded.

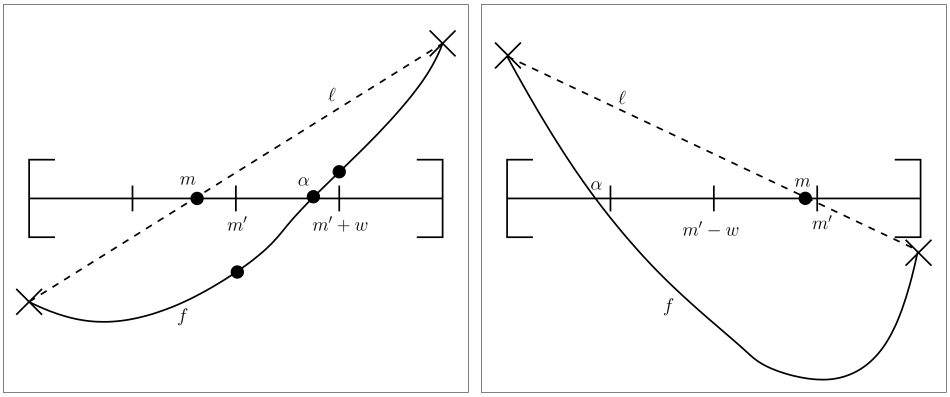


Figure 2.10. A successful (left), and a non-successful (right) qir call for $N = 4$.

Definition 2.5.2 (successful/failing qir call). A qir call $(J, N_2) \leftarrow \text{QIR}(f,I,N_1)$ *succeeds* if $J \subsetneq I$, and it *fails* if $J = I$. Equivalently, the qir calls succeeds, if and only if $N_2 > N_1$.

Clearly, a successful qir call at least halves the interval; more precisely, a successful qir call for N has the same effect as performing $\log N$ bisection steps. We show next that one qir call (successful or not) is not more costly than one bisection regarding their bit complexities.

Proposition 2.5.3. *Let $f \in \mathbb{Z}[t]$ be of magnitude (n, τ), and let I be an isolating standard interval of bitsize σ. Using Algorithm 2.12, one bisection step requires $O(n)$ arithmetic operations in $\mathbb{Z}$. The bit complexity is $O(nM(\tau + n\sigma))$.*

Proof. The arithmetic complexity follows because one has to evaluate f at c, d, m', and $m' \pm w$, and perform another constant number of arithmetic operations. The bitsize of m' and $m' \pm w$ is bounded by $O(\sigma + \log N)$.

It is easy to see that $\log N \in O(\sigma)$, assuming that the qir is initially started with $N = 4$: if a qir call with $N > 4$ subintervals is started, there must have been a successful qir call for $\sqrt{N}$. Thus, the bitsize of either c or d must be at least $\log \sqrt{N}$. So, $\sigma \geq \log \sqrt{N} = \frac{1}{2} \log N$. The bit complexity follows with Lemma 2.4.10 (univariate polynomial evaluation). $\square$

2.5.2. Strong root isolation

We have presented two ways to refine the isolating interval of an algebraic number. When only considering one refinement step, it is not clear which method is preferable – bisection halves the interval in each step; the qir method can refine much better in one step, but there is also the possibility of failure.

In this section, which summarizes the results of [Ker09b], we argue that the qir method is the superior choice for root refinement, if one considers a sequence of refinement steps. For that, we look at a variant of the refinement problem, which we call *strong root iso-*

Algorithm 2.13. Strong root isolation

INPUT: $f \in \mathbb{Z}[t]$ square-free, of magnitude (n, τ), $\varepsilon > 0$
OUTPUT: $I_1, \dots, I_m$ isolating with $\mathrm{w}(I_i) < \varepsilon$

procedure STRONG_ISOLATE(f)
  $I_1, \dots, I_m \leftarrow$ SOLVE(f)
  for $k \in \{1, \dots, m\}$ **do**
    $N \leftarrow 4$
    while $\mathrm{w}(I_i) > \varepsilon$ **do** $(I_i, N) \leftarrow$ QIR(f, I_i, N)
  end for
  return $I_1, \dots, I_m$
end procedure

lation:[17] *Given some $f \in \mathbb{Z}[t]$ and $\varepsilon > 0$, compute isolating intervals for the roots of f, each of a width of less than ε.* Note that a sequence of refinement steps is a more meaningful scenario to look at than just one single refinement step, because one typically keeps on refining the interval until a certain property of the algebraic number, for instance, its (non-zero) sign, can be deduced (e.g., compare Algorithms 2.17 and 2.18).

Why do we consider all roots of a polynomial at once, instead of fixing one isolating interval? The reason is that the analysis will depend on quantities like $|\alpha|$ or $|f'(\alpha)|$, and considering all roots at once allows us to bound these quantities by algebraic expressions like the Mahler measure of f, or the resultant of f and f'. This will lead to an improved complexity bound compared to the approach of fixing one interval, and multiply the bound with n when considering all real roots.

Our algorithm for strong root isolation is so simple that we can state it right away – isolate the real roots, and iteratively call the qir method for each interval until it is smaller than ε. The pseudo-code is given in Algorithm 2.13.

The analysis of this algorithm requires more work and appears to be new in the form we present it (in the abstract of the original poster [Abb06], it is only briefly sketched out that quadratic convergence finally occurs). We decided to assume fast integer multiplication for the analysis (and in fact, for the remainder of this thesis), which means that we set $M(n) = O(n \log n 2^{O(\log^* n)})$. For simplicity, we do not consider logarithmic factors in n or τ in the analysis and write $\tilde{O}$ to denote such a complexity bound. We also assume fast Taylor shifts, which means that, we set $T(n, \tau) = \tilde{O}(n\tau)$. There are two reasons why this simplification is made: first of all, the expressions occurring in the complexity bounds become hardly manageable when the cost of multiplication is left open. Secondly, different choices of multiplication complexity lead to different analysis techniques to derive an optimized bound; our method yields a good worst-case complexity assuming fast integer arithmetic, but it would not be appropriate when assuming, for instance, classic arithmetic.

For convenience, we set $L_\varepsilon := \log \frac{1}{\varepsilon}$. We will show the following statement. For a polynomial of magnitude (n, τ), strong root isolation with f and ε has a bit complexity of

$$\tilde{O}\left(n^4 \tau^2 + n^3 L_\varepsilon\right).$$

To put this result into perspective, we first analyze the complexity of Algorithm 2.13, if bisection was chosen instead of qir. Then, we need up to $O(\tau + L_\varepsilon)$ bisections for each

[17] This term was introduced by Chee Yap, according to a remark by Vikram Sharma.

interval. This causes an arithmetic complexity of $O(n^2(\tau + L_\varepsilon))$ (one factor n comes from the number of intervals, one from the cost of one bisection). During this process, the bitsize σ of the intervals becomes as large as $O(\tau + L_\varepsilon)$ as well. Thus, the bit complexity of all refinement steps is $O(n^2(\tau + L_\varepsilon)M(n(\tau + L_\varepsilon)))$, according to Proposition 2.5.3. Adding the cost of root isolation (Theorem 2.4.38), we obtain a total complexity of

$$O(T(n)n^3\tau^2 + n^2(\tau + L_\varepsilon)M(n(\tau + L_\varepsilon))) = \tilde{O}(n^4\tau^2 + n^3(\tau + L_\varepsilon)^2) = \tilde{O}(n^4\tau^2 + n^3L_\varepsilon^2).$$

With the qir method, the quantity L_ε only appears linearly in the bound.

The reader might wonder at this point why we are not using a more prominent algorithm like the famous *Newton's method* instead of the qir method. A problem with Newton's method lies in the choice of a starting value – an unfortunate choice leads to a divergent sequence. A solution is to initially perform bisections to produce an interval where convergence of Newton's method is guaranteed, and then to switch to Newton iteration manually. However, this manual switch depends on theoretical worst-case bounds for valid starting values of Newton's method, thus more bisections than necessary are performed in the average case. The qir method, in contrast, switches adaptively as soon as possible, independently of the worst-case bounds that are introduced only for the analysis.

Dekker [Dek69] presented a method which, similarly to the qir, combines bisections and the secant method. Brent [Bre73] combines Dekker's method with inverse quadratic interpolation. Superlinear convergence can also be guaranteed for this method. However, a problem with Dekker's approach is the growth in the bitsize of the iteration values – it is unclear how to choose a suitable working precision in each substep to avoid too large coefficients while still guaranteeing fast convergence. The same holds true for Brent's method, and additionally, an analysis seems to be even more involved since it adds even more ingredients to Dekker's method. The qir method guarantees a minimal growth in the bitsizes, since all intervals are standard intervals; this means that the bitsize of the boundaries is proportional to the interval width, which is the best one can hope for.

Definition 2.5.4 (qir sequence). Let α be a root of f for which Step 2 of Algorithm 2.13 returned the isolating interval I_0. The *qir sequence* $(s_0, \ldots, s_n)$ for α, is defined as

$$s_0 := (I_0, 4) \qquad s_i := (I_i, N_i) := \text{QIR}(f, I_{i-1}, N_{i-1}) \quad \text{for } i \geq 1$$

where I_n is the first index such that $w(I_n) \leq \varepsilon$. We say that $s_{i-1} \overset{\text{QIR}}{\to} s_i$ *succeeds* if $\text{QIR}(f, I_{i-1}, N_{i-1})$ succeeds, and that $s_{i-1} \overset{\text{QIR}}{\to} s_i$ *fails* otherwise.

The key to the improvement when using qir instead of bisection is that the refinement process develops quadratic convergence behavior, because from a certain point on, all qir calls succeed. The quantity M_α as defined next will be important, because it constitutes the bound for which quadratic convergence can be guaranteed.

Lemma 2.5.5. *Let $\alpha \in \mathbb{C}$ be a root of f, and let f be of magnitude (n, τ). We define*

$$M_\alpha := \frac{|f'(\alpha)|}{2en^3 2^\tau \max\{|\alpha|, 1\}^{n-1}}$$

(with $e \approx 2.71$). It holds true that:

1. $0 < M_\alpha \leq \frac{1}{n}$

2. $M_\alpha < \frac{|f'(\alpha)|}{2|f''(\mu)|}$, *if* $\mu \in \mathbb{C}$ *such that* $|\alpha - \mu| < M_\alpha$.

Proof. We bound $|f'(\alpha)|$ from above by

$$|f'(\alpha)| = |\sum_{i=1}^{n} i a_i \alpha^{i-1}| \le n2^\tau \sum_{i=0}^{n-1} \max\{|\alpha|, 1\}^i \le n2^\tau n \max\{|\alpha|, 1\}^{n-1},$$

which proves the first claim. For the second, we bound $|f''(\mu)|$ from above:

$$\begin{aligned}
|f''(\mu)| &= |\sum_{i=2}^{n} i(i-1) a_i \mu^{i-2}| \\
&\le n^2 2^\tau \sum_{i=0}^{n-2} \max\{|\mu|, 1\}^i \\
&\le n^2 2^\tau \sum_{i=0}^{n-2} ((1 + M_\alpha) \max\{|\alpha|, 1\})^i \\
&\le n^3 2^\tau (1 + M_\alpha)^{n-2} \max\{|\alpha|, 1\}^{n-2} \\
&< n^3 2^\tau \underbrace{(1 + \frac{1}{n})^n}_{<e} \max\{|\alpha|, 1\}^{n-1}.
\end{aligned}$$

This proves that

$$\frac{|f'(\alpha)|}{2|f''(\mu)|} > \frac{|f'(\alpha)|}{2e \cdot n^3 2^\tau \max\{|\alpha|, 1\}^{n-1}} = M_\alpha. \quad \square$$

We can now split the qir sequence into two subsequences.

Definition 2.5.6 (initial/quadratic sequence). Let $(s_0, \dots, s_n)$ be the qir sequence for α. Let k be the minimal index such that $s_k = (I_k, N_k) \overset{\text{QIR}}{\to} s_{k+1}$ succeeds, and $\mathrm{w}(I_k) \le M_\alpha$. We call the sequence $(s_0, \dots, s_k)$ the *initial sequence*, and $(s_k, \dots, s_p)$ the *quadratic sequence*.

In the next two subsections, we will bound the cost of the initial sequence and the quadratic sequence separately.

Cost of the initial sequence

Regarding the complexity, we show that refining to width M_α using qir is at least not worse than using bisections.

Lemma 2.5.7. *Let I be an isolating interval for α, returned by Algorithm 2.10. The cost of the initial sequence of α is bounded by*

$$O(n(\sigma + \log \frac{1}{M_\alpha}) M(n(\sigma + \log \frac{1}{M_\alpha}))) = \tilde{O}(n^2(\tau + \log \frac{1}{M_\alpha})^2).$$

Proof. Let n_q be the number of qir calls until I is refined such that $w(I) < M_\alpha$. Likewise, let n_b be the number of bisections that are needed until the initial interval I is refined to size M_α. Note that $n_b = O(\sigma + \log \frac{1}{M_\alpha})$.

A successful qir call for some $N = 2^{2^i}$ yields the same result as 2^i bisections and can only cause up to $i + 1$ subsequent failing qir calls before the next successful qir call.

With that argument, it follows that $n_q \leq 2n_b$, so the number of function calls is at most doubled when qir is used instead of bisection. Consequently, the arithmetic complexity is $O(n(\tau + \log \frac{1}{M_\alpha}))$.

To bound the the bit complexity, let N_e be the value of N in the last qir call of the initial sequence. Clearly, $\log N_e \leq 2n_b$, since otherwise, the preceding qir call would have yielded as much accuracy as $\log \sqrt{N_e} > n_b$ bisections and the method would have terminated earlier. Hence, it follows that the width of the final interval is at least $\frac{M_\alpha}{N_e}$ and therefore, the interval boundaries have a bitsize of at most

$$\log \frac{N_e}{M_\alpha} \leq 2n_b + \log \frac{1}{M_\alpha} = O(\tau + \log \frac{1}{M_\alpha}).$$

Therefore, the bitsizes of the qir calls are bounded by $O(n(\tau + \log \frac{1}{M_\alpha}))$, which proves the claim. □

It remains to bound the quantity $\log \frac{1}{M_\alpha}$. We do this simultaneously for all real roots of the polynomial, according to the following theorem.

Theorem 2.5.8. *Let $\alpha_1, \ldots, \alpha_m$ be the real roots of f. Then,*

$$\sum_{i=0}^{m} \log \frac{1}{M_{\alpha_i}} = O(n(\tau + \log n)).$$

Proof. Recall that $M_\alpha < 1$ for each root α, including the non-real roots, which we denote by $\alpha_{m+1}, \ldots, \alpha_n$. Therefore, $\log \frac{1}{M_\alpha} > 0$, and we can bound:

$$\begin{aligned}
\sum_{i=0}^{m} \log \frac{1}{M_{\alpha_i}} &\leq \sum_{i=0}^{n} \log \frac{1}{M_{\alpha_i}} \\
&= \log \frac{\prod_{i=1}^{n} 2e \cdot n^3 2^\tau \max\{|\alpha_i|, 1\}^{n-1}}{|\prod_{i=1}^{n} f'(\alpha_i)|} \\
&= n \log(2e) + 3n \log n + n\tau + (n-1) \log \prod_{i=1}^{n} \max\{|\alpha_i|, 1\} - \log |\prod_{i=1}^{n} f'(\alpha_i)|.
\end{aligned}$$

The first product can be written in terms of the Mahler measure:

$$\log \prod_{i=1}^{n} \max\{|\alpha_i|, 1\} = \log \left(\frac{1}{|a_n|} \mathrm{Mea}(f) \right) \leq \log \mathrm{Mea}(f) \leq \log(\sqrt{n+1} \cdot 2^\tau) = O(\log n + \tau).$$

For the second product, recall from Theorem 2.3.6 that the resultant of f and f' can be expressed as

$$\mathrm{res}(f, f') = a_n^{n-1} \prod_{i=1}^{n} f'(\alpha_i).$$

Consequently,

$$-\log |\prod_{i=1}^{n} f'(\alpha_i)| = -\log \frac{|\mathrm{res}(f, f')|}{|a_n^{n-1}|} = \log |a_n^{n-1}| - \log |\underbrace{\mathrm{res}(f, f')}_{\geq 1}| < (n-1) \log |a_n| = O(n\tau).$$

It follows that

$$\begin{aligned}
\sum_{i=0}^{m} \log \frac{1}{M_{\alpha_i}} &\leq n \log 2e + 3n \log n + n\tau + (n-1) \underbrace{\log \prod_{i=1}^{n} \max\{|\alpha_i|, 1\}}_{=O(\log n + \tau)} - \underbrace{\log |\prod_{i=1}^{n} f'(\alpha_i)|}_{=O(n\tau)} \\
&= O(n(\tau + \log n)). \quad \square
\end{aligned}$$

Corollary 2.5.9. *The total computation cost for all initial sequences for $\alpha_1, \ldots, \alpha_s$ is*

$$\tilde{O}(n^4\tau^2).$$

Proof. Combining Lemma 2.5.7 and Theorem 2.5.8, we get total costs of

$$\tilde{O}(\sum_{i=1}^{s} n^2(\tau + \log \frac{1}{M_{\alpha_i}})^2) = \tilde{O}(n^3\tau^2 + n^2(\sum_{i=1}^{s} \log \frac{1}{M_{\alpha_i}})^2) = \tilde{O}(n^4\tau^2). \quad \square$$

Cost of the quadratic sequence

In the initial sequence, we have assumed that the qir sequence behaves roughly as the bisection method does. However, as soon as the isolating interval becomes smaller than M_α, we can prove that N is doubled in almost every step, and so, the precision of the interval is basically doubled in each step. We start with a simple criterion to guarantee success of a qir call.

Proposition 2.5.10. *Let $I = [c, d]$ be an isolating interval of α, with $\mathrm{w}(I) = \delta$, and consider the qir call* QIR*(f,I,N) for some N. Let $m := c + \frac{f(c)}{f(c)-f(d)}(c - d)$ as defined in the qir method. If $|m - \alpha| < \frac{\delta}{2N}$, the qir call succeeds.*

Proof. Let J be the subinterval that contains α, and J' be the subinterval that contains m. If $J = J'$, then one of the endpoints of J' is chosen as m', so the qir call succeeds. If $J \neq J'$, they must be adjacent, since otherwise, $|m - \alpha| > \frac{\delta}{N}$. W. l. o. g., assume that $m < \alpha$, otherwise the argument is symmetric. It follows that m must be in the right half of J, because otherwise $|m - \alpha| > \frac{\delta}{2N}$. Thus, m' is chosen as the right endpoint of J', which is the left endpoint of J. Therefore, the qir call succeeds. $\square$

We need to investigate the distance between the interpolation point m and the root α. The next theorem shows that this distance depends quadratically on the width of the isolating interval once it is smaller than M_α. A similar result can be used to show quadratic convergence of Newton's method [Yap00, §6.11].

Theorem 2.5.11. *Let (c, d) be an isolating interval for α, and let $\delta := d - c$ be its width. If $\delta < M_\alpha$, then $|m - \alpha| < \frac{\delta^2}{2M_\alpha}$.*

Proof. Consider the Taylor expansion of f at α. For a given x, we have

$$f(x) = f'(\alpha)(x - \alpha) + \frac{1}{2}f''(\tilde{\alpha})(x - \alpha)^2$$

with some $\tilde{\alpha} \in [x, \alpha]$ or $[\alpha, x]$. Thus, we have

$$\begin{aligned}
& |m-\alpha| \\
= & \left|\frac{f(d)(c-\alpha)-f(c)(d-\alpha)}{f(d)-f(c)}\right| \\
= & \left|\frac{(\frac{1}{2}f''(\tilde{\alpha}_1)(d-\alpha)^2+f'(\alpha)(d-\alpha))(c-\alpha)-(\frac{1}{2}f''(\tilde{\alpha}_2)(c-\alpha)^2-f'(\alpha)(c-\alpha))(d-\alpha)}{f(d)-f(c)}\right| \\
= & \left|\frac{\frac{1}{2}(f''(\tilde{\alpha}_1)(d-\alpha)^2(c-\alpha)-f''(\tilde{\alpha}_2)(c-\alpha)^2(d-\alpha))}{f(d)-f(c)}\right| \\
= & \frac{1}{2}|d-\alpha||c-\alpha|\left|\frac{f''(\tilde{\alpha}_1)(d-\alpha)-f''(\tilde{\alpha}_2)(c-\alpha)}{f(d)-f(c)}\right| \\
\leq & \frac{1}{2}\delta^2\cdot\frac{|f''(\tilde{\alpha}_1)|(d-\alpha)+|f''(\tilde{\alpha}_2)|(\alpha-c)}{f(d)-f(c)} \\
\leq & \frac{1}{2}\max\{|f''(\tilde{\alpha}_1)|,|f''(\tilde{\alpha}_2)|\}\delta^2\frac{(d-\alpha)+(\alpha-c)}{f(d)-f(c)} \\
= & \frac{\max\{|f''(\tilde{\alpha}_1)|,|f''(\tilde{\alpha}_2)|\}\delta^2}{2f'(\nu)}
\end{aligned}$$

for some $\nu \in [c, d]$. The Taylor expansion of f' yields

$$f'(\nu) = f'(\alpha) + f''(\tilde{\nu})(\nu - \alpha)$$

for some $\tilde{\nu} \in [c, d]$. Since $\delta \leq M_\alpha$, we have

$$|f''(\tilde{\nu})(\nu-\alpha)| \leq |f''(\tilde{\nu})|M_\alpha \leq \frac{1}{2}|f'(\alpha)|,$$

according to Lemma 2.5.5. Therefore, it holds that $f'(\nu) > \frac{1}{2}f'(\alpha)$, and it follows that

$$|m-\alpha| \leq \frac{\max\{|f''(\tilde{\alpha}_1)|,|f''(\tilde{\alpha}_2)|\}\delta^2}{|f'(\alpha)|} \leq \frac{\delta^2}{2\frac{|f'(\alpha)|}{2\max\{|f''(\tilde{\alpha}_1)|,|f''(\tilde{\alpha}_2)|\}}} < \frac{\delta^2}{2M_\alpha}. \quad \square$$

We apply this theorem to the quadratic sequence.

Corollary 2.5.12. *Let I_j be an isolating interval for α of width $\delta_j \leq \frac{1}{N_j}M_\alpha$. Then, each call of the qir sequence $(I_j, N_j) \overset{\text{QIR}}{\to} (I_{j+1}, N_{j+1}) \overset{\text{QIR}}{\to} \ldots$ succeeds.*

Proof. We do induction on i. Assume (for $i \geq 0$) that the first i calls succeeded. Then, it is easily shown that $\delta_{j+i} := w(I_{j+i}) = \frac{N_j\delta_j}{N_{j+i}} < \frac{M_\alpha}{N_{j+i}}$ (by another induction, and exploiting the fact that $N_{j+i}^2 = N_{j+i+1}$). Using Theorem 2.5.11, we have

$$|m-\alpha| \leq \delta_{j+i}^2\frac{1}{2M_\alpha} \leq \delta_{j+i}\frac{M_\alpha}{N_{j+i}}\frac{1}{2M_\alpha} = \frac{1}{2}\frac{\delta_{j+i}}{N_{j+i}}.$$

By Proposition 2.5.10, this is enough to guarantee success for the qir method. $\square$

Corollary 2.5.13. *In the quadratic sequence, there is at most one failing qir call.*

Proof. Let $(I_i, N_i) \overset{\text{QIR}}{\to} (I_{i+1}, N_{i+1})$ be the first failing qir call in the quadratic sequence. Since the quadratic sequence starts with a successful qir call, the predecessor $(I_{i-1}, N_{i-1}) \overset{\text{QIR}}{\to} (I_i, N_i)$ is also part of the quadratic sequence and succeeds. Thus, we have the sequence

$$(I_{i-1}, N_{i-1}) \overset{\text{Sucess}}{\overset{\text{QIR}}{\to}} (I_i, N_i) \overset{\text{Fail}}{\overset{\text{QIR}}{\to}} (I_{i+1}, N_{i+1}) \overset{\text{QIR}}{\to} \dots.$$

One observes that $w(I_{i+1}) = w(I_i) = \frac{w(I_{i-1})}{N_{i-1}} \leq \frac{M_\alpha}{N_{i-1}}$, and $N_{i+1} = \sqrt{N_i} = \sqrt{N_{i-1}^2} = N_{i-1}$. By Corollary 2.5.12, all further qir calls succeed. □

If the quadratic sequence starts with a bisection (i.e., $N = 2$ initially), no failing qir call occurs. Otherwise, the single failing step is due to the fact that the quadratic sequence might start with a too big value of N, just because the algorithm was "too lucky" during the initial sequence.

Let $(I_{i-1}, N_{i-1}) \overset{\text{QIR}}{\to} (I_i, N_i)$ be the failing qir call in the quadratic sequence. Since $w(I_{i+k}) = \frac{N_i w(I_i)}{N_{i+k}}$ by the proof of Corollary 2.5.13, it follows that

$$w(I_{i+k+1}) = \frac{w(I_{i+k})^2}{N_i \cdot w(I_i)}$$

for any $k \geq 0$. This means that the interval width decreases quadratically in each step (up to the constant $N_i \cdot w(I_i)$), which ultimately justifies the term "quadratic" in the quadratic interval refinement method (the idea of our exposition has already been sketched out in Abbott's original work [Abb06]).

Lemma 2.5.14. *The number of bit operations in the quadratic sequence of a root α is bounded by*

$$\tilde{O}(n^2 \log L(\sigma + \log \frac{1}{M_\alpha}) + n^2 L).$$

Proof. By Corollary 2.5.13, the quadratic sequence consists of at most $\log L + 1$ qir calls, since N is doubled in each step, except the possible failing step. The bitsize in the first qir call of the sequence is $O(n(\sigma + \log \frac{1}{M_\alpha}))$ and increases by at most 2^i after the i-th iteration. Therefore, the complexity of the quadratic sequence is given by

$$O\left(\sum_{i=1}^{\log L+1} n \cdot M(n(\sigma + \log \frac{1}{M_\alpha} + 2^i))\right) = \tilde{O}\left(n^2 \sum_{i=1}^{\log L+1} \sigma + \log \frac{1}{M_\alpha} + 2^i\right)$$

$$= \tilde{O}\left(n^2 \log L(\sigma + \log \frac{1}{M_\alpha}) + n^2 \sum_{i=1}^{\log L+1} 2^i)\right) = \tilde{O}(n^2 \log L(\sigma + \log \frac{1}{M_\alpha}) + n^2 L). \quad □$$

Corollary 2.5.15. *The total cost of all quadratic sequences for the real roots $\alpha_1, \dots, \alpha_s$ of f is bounded by*

$$\tilde{O}(n^3 \sigma \log L + n^3 L).$$

Proof. We combine Lemma 2.5.14 and Theorem 2.5.8 to obtain

$$\sum_{i=1}^{s} \tilde{O}(n^2 \log L(\sigma + \log \frac{1}{M_\alpha}) + n^2 L) = \tilde{O}(n^3 \sigma \log L + n^2 \log L \underbrace{\sum_{i=1}^{s} \log \frac{1}{M_\alpha}}_{=O(n(\sigma + \log n))} + n^3 L). \quad □$$

Algorithm 2.14. Finding intermediate values

INPUT: $f \in \mathbb{Z}[t]$ square-free, of magnitude (n, τ)

OUTPUT: $q_0, \ldots, q_m \in \mathbb{Q}$ such that each interval (q_i, q_{i+1}) contains precisely one root of f.

```
procedure INTERMEDIATE(f)
    I_1, ..., I_m ← SOLVE(f)
    q_0 ← −2^(τ+1), q_m ← 2^(τ+1)
    for k ∈ {1, ..., m − 1} do
        while I_i and I_{i+1} have a common endpoint, do refine both intervals using QIR
        d_i ← right endpoint of I_i, c_{i+1} ← left endpoint of I_{i+1}
        q_i ← (d_i + c_{i+1}) / 2
    end for
    return q_0, ..., q_m
end procedure
```

We now can finally prove the main result:

Theorem 2.5.16 (strong root isolation). *Assuming fast integer arithmetic and fast Taylor shifts, the strong root isolation problem for f (square-free) and ε has a bit complexity of*

$$\tilde{O}(n^4\tau^2 + n^3 L_\varepsilon)$$

with $L_\varepsilon = \log \frac{1}{\varepsilon}$.

Proof. The costs are determined by the sum of

1. the real root isolation which is $\tilde{O}(n^4\tau^2)$ (Theorem 2.4.38),
2. the cost of the initial sequence which is $\tilde{O}(n^4\tau^2)$ (Corollary 2.5.9), and
3. the cost of the quadratic sequence which is $\tilde{O}(n^3 \log L\tau + n^3 L)$ (Corollary 2.5.15).

We have to argue why the term $n^3\tau \log L$ never becomes dominant, and can thus be removed. If $p^4\sigma^2$ were dominated by $p^3\sigma \log L$, $\log L$ would dominate $p\sigma$, and in particular L would dominate 2^σ. If also $p^3 L$ were dominated by $p^3\sigma \log L$, then $\frac{L}{\log L}$ would be dominated by σ, and so L would be dominated by $\sigma^{1+\gamma}$ for any $\gamma > 0$. □

We will use Theorem 2.5.16 frequently in this work since we will need close approximations of algebraic numbers in numerous situations. As a first application, consider the following problem. For a (square-free) polynomial f, we want to find *intermediate* values for the roots, which means that if the real roots are $\alpha_1, \ldots, \alpha_m$, then we want to find $q_0, \ldots, q_m \in \mathbb{Q}$ such that $q_0 < \alpha_1 < q_1 < \ldots < \alpha_m < q_m$. In most cases, such intermediate values can be computed from the output of the Descartes method; if $I_i = (c_i, d_i)$ and $I_{i+1} = (c_{i+1}, d_{i+1})$ are consecutive isolating intervals, then q_i can be set to $\frac{d_i + c_{i+1}}{2}$. But what if $I_i = [c, c]$, and $I_{i+1} = (c, d)$? Then, I_{i+1} must be refined until its left boundary changes.

Theorem 2.5.17. *The bit complexity of Algorithm 2.14 is bounded by*

$$\tilde{O}(n^4\tau^2).$$

All intermediate values have a maximum bitsize of $O(\tau + L_f) = O(n(\tau + \log n))$, and the first and the last intermediate values each have a bitsize of $O(\tau)$.

Proof. Set $\varepsilon := \mathrm{sep}(f)/4$. The isolating intervals do no longer touch at their boundaries when all of them are refined to width ε at the latest. In this situation, Algorithm 2.14 will compute the intermediate value. Refining all intervals to size ε has bit complexity

$$\tilde{O}(n^4\tau^2 + n^3(L_\varepsilon + \tau \log L_\varepsilon))$$

and by the fact that $L_\varepsilon = 2 + L_f = O(n(\tau + \log n))$, this simplifies to $O(n^4\tau^2)$. ☐

The proof has shown more. Refining all isolating intervals of the polynomial to the size of its separation bound is asymptotically not more expensive than the root isolation itself. This appears surprising, and strongly depends on the quadratic convergence of the qir method – if bisection were used, the refinements would increase the complexity.

The assumption that all intervals are refined to width $\mathrm{sep}(f)/4$ is made *only* for the worst-case analysis. Note that Algorithm 2.14 does not ,in general, require this; it terminates as soon as no intervals share an endpoint, which can happen much more quickly. In this sense, the algorithm is adaptive to the given problem, and does not depend on worst-case bounds of the root separation. This property is characteristic for all algorithms that will follow in this chapter.

2.5.3. Comparing algebraic numbers

The next problem we look at is comparing two algebraic numbers $\alpha = (f_1, I_1)$ and $\beta = (f_2, I_2)$, both in isolating interval representation (specifically, both f_1 and f_2 are square-free). We use a two-step approach for this. First, we check for equality by computing $\gcd(f_1, f_2)$, and checking whether a sign change in $I_1 \cap I_2$ occurs. If α and β are not equal, both isolating intervals are refined until disjointness in the second step.

Theorem 2.5.18. *Algorithm 2.15 correctly compares algebraic numbers. For f_1, f_2 of magnitude (n, τ), the procedure* EQUAL *has a bit complexity of $\tilde{O}(n^3(n+\tau))$.* DIST_COMPARE *and* COMPARE *are of bit complexity $\tilde{O}(n^4\tau^2)$.*

Proof. We argue first about the correctness of the procedure EQUAL. Assume that $\alpha = \beta$. Then, α is also a root of $h = \gcd(f_1, f_2)$, the gcd of the defining polynomials (see Algorithm 2.15), and the interval $I_1 \cap I_2$ is isolating for α (because an additional root would violate the assumption that I_1 and I_2 are isolating). Thus, since h is square-free as well, the sign at the interval boundaries differs. Vice versa, if the sign differs, then f_1 and f_2 have (at least) one common root in $I_1 \cap I_2$ and this is only possible if $\alpha = \beta$. The correctness of the procedures DIST_COMPARE and COMPARE is obvious.

For the bit complexity of EQUAL, note that the gcd computation is within $O(n^2 M(n(\tau + \log n))) = \tilde{O}(n^3\tau)$ according to Theorem 2.4.19, and h is of magnitude $(n, n+\tau)$. h is now evaluated at the interval boundaries of I, which are of bitsize $O(n(\tau + \log n)) = \tilde{O}(n\tau)$. This requires another $\tilde{O}(n^3\tau)$ bit operations according to Lemma 2.4.10 (evaluation of univariate polynomials).

For the bit complexity of DIST_COMPARE, consider the separation of $f_1 f_2$. As before, it holds that

$$L_{fg} = \frac{1}{\mathrm{sep}(f_1 \cdot f_2)} = O(2n(2\tau + \log 2n)) = O(n(\tau + \log n)).$$

Algorithm 2.15. Comparing algebraic numbers

INPUT: $(f_1, I_1), (f_2, I_2)$ with $f_1, f_2 \in \mathbb{Z}[t]$ square-free and of magnitude (n, τ), I_1 isolating interval for $\alpha \in \mathbb{R}$, I_2 for $\beta \in \mathbb{R}$
OUTPUT: $\alpha = \beta$

procedure EQUAL($(f_1, I_1), (f_2, I_2)$)
  $I \leftarrow I_1 \cap I_2$
  if $I = \emptyset$ **return** false
  $h \leftarrow \gcd(f_1, f_2)$
  if $I = [c, c]$ **return** $h(c) = 0$ ▷ Otherwise, I=$[c, d]$
  return sign $h(c) \cdot$ sign $h(d) < 0$
end procedure

INPUT: $\alpha = (f_1, I_1), \beta = (f_2, I_2)$ as above, $\alpha \neq \beta$
OUTPUT: $\alpha < \beta$ (LESS) or $\alpha > \beta$ (GREATER)

procedure DIST_COMPARE($(f_1, I_1), (f_2, I_2)$)
  while $I_1 \cap I_2 \neq \emptyset$ **do**
    refine I_1 and I_2 using QIR
  end while
  if $I_1 < I_2$ **return** LESS **else return** GREATER
end procedure

INPUT: $\alpha = (f_1, I_1), \beta = (f_2, I_2)$ as above
OUTPUT: $\alpha < \beta$ (LESS) or $\alpha = \beta$ (EQUAL) or $\alpha > \beta$ (GREATER)

procedure COMPARE($(f_1, I_1), (f_2, I_2)$)
  if EQUAL($(f_1, I_1), (f_2, I_2)$), **return** EQUAL
  else return DIST_COMPARE($(f_1, I_1), (f_2, I_2)$)
end procedure

Once both intervals are refined to size $\frac{1}{4}\mathrm{sep}(f_1 f_2)$, they must be disjoint, and the bit complexity of this step is bounded by $\tilde{O}(n^4\tau^2)$ by Theorem 2.5.16. The bit complexity of COMPARE follows immediately. □

As we can see from the proof, the (theoretical) bottleneck in the comparison method is the refinement step for non-equal algebraic numbers. If we instead compare all roots of two square-free polynomials f_1 and f_2, we obtain the same asymptotic bound. By comparing all roots, we understand that the isolating intervals of two polynomials f_1 and f_2 are merged into one sorted list of isolating intervals. This could also be done by real root isolation for $f \cdot g$, but Algorithm 2.16 is faster in practice, and computes the additional information whether a root belongs to f_1, to f_2, or to both.

Theorem 2.5.19. *Algorithm 2.16 merges the roots of f_1 and f_2 within bit complexity $\tilde{O}(n^4\tau^2)$.*

Proof. The initial calls of SOLVE have a bit complexity of $\tilde{O}(n^4\tau^2)$. The only crucial operations afterwards are the (up to) $m_1 + m_2 \leq 2n$ calls of COMPARE: Calling EQUAL so often yields at most $\tilde{O}(n^4\tau)$ operations, which matches the complexity bound. During the DIST_COMPARE calls, each interval of f_1 and of f_2 is refined to a maximal precision of $L_{fg} = O(n(\tau + \log n))$ with the same argument as in the proof of Theorem 2.5.18, and

Algorithm 2.16. Merging algebraic numbers

INPUT: $f_1, f_2 \in \mathbb{Z}[t]$ square-free and of magnitude (n, τ)

OUTPUT: A list of pairs $(I_1, b_1), \ldots, (I_s, b_s)$, where $I_1, \ldots, I_s$ are disjoint isolating intervals for all roots of fg, and $b_i \in \{\text{FIRST,SECOND,BOTH}\}$ is a flag denoting whether the root in I_I is a root of f_1 (FIRST), a root of f_2 (SECOND), or of both (BOTH)

procedure MERGE_ROOTS(f_1, f_2)
  $I_1^{(1)}, \ldots, I_{m_1}^{(1)} \leftarrow$ SOLVE(f_1) ▷ m_1 is the number of real roots of f_1
  $I_1^{(2)}, \ldots, I_{m_2}^{(2)} \leftarrow$ SOLVE(f_2) ▷ m_2 is the number of real roots of f_2
  $i_1 \leftarrow 1, i_2 \leftarrow 1, j \leftarrow 1$
  while $i_1 \neq m_1$ or $i_2 \neq m_2$ **do**
    $s \leftarrow$COMPARE$((f_1, I_{i_1}^{(1)}), (f_2, I_{i_2}^{(2)}))$
    if $s = LESS$, $J_j \leftarrow (I_{i_1}^{(1)}, FIRST)$, $i_1 \leftarrow i_1 + 1$
    if $s = GREATER$, $J_j \leftarrow (I_{i_2}^{(2)}, SECOND)$, $i_2 \leftarrow i_2 + 1$
    if $s = EQUAL$, $J_j \leftarrow (I_{i_1}^{(1)}, BOTH)$, $i_1 \leftarrow i_1 + 1, i_2 \leftarrow i_2 + 1$
  end while
  return $J_0, \ldots, J_j$
end procedure

refining all isolating intervals to this bound can be done with $\tilde{O}(n^4\tau^2)$ bit operations using Theorem 2.5.16. ☐

2.5.4. Interval arithmetic

The reader might wonder why we so far have not discussed basic arithmetic operations for algebraic numbers, such as addition or multiplication. Indeed, it is well known that if α and β are algebraic, both their sum and their product are algebraic. There are also algorithms known for constructing polynomials that contain the sum or the product as one of their roots (e.g., [Loo82a] [Yap00, Lemma 6.16]). This construction requires resultant computations and is known to be expensive in practice. Moreover, in our applications, we will not need to compute an isolating interval representation of a sum, or a product, of algebraic numbers – what we will need, however, are *approximated* sums and products.

For that, we employ the well-known technique of *interval arithmetic*. There are several textbooks dedicated to this discipline [Moo79] [AH83], applications in computer graphics [SF91] and in *cad* computation [CJK02] have been already considered. The methods that we require are very elementary and should be understandable without further knowledge of that area.

We define the addition, multiplication, and division of intervals. We restrict our discussion to closed intervals in this treatment; for interval arithmetic on (half-)open intervals, one can simply pass to their closure.

$$\begin{aligned}
[a,b] \oplus [c,d] &= [a+c, b+d] \\
[a,b] \ominus [c,d] &= [a-d, b-c] \\
[a,b] \odot [c,d] &= [\min\{ac, bc, ad, bf\}, \max\{ac, bc, ad, bd\}] \\
[a,b] \oslash [c,d] &= [\min\{\frac{a}{c}, \frac{b}{c}, \frac{a}{d}, \frac{b}{d}\}, \max\{\frac{a}{c}, \frac{b}{c}, \frac{a}{d}, \frac{b}{d}\}]
\end{aligned}$$

The division rule requires that $0 \notin [c, d]$. Operations with real numbers are possible simply by interpreting $r \in \mathbb{R}$ as the interval $[r, r]$. It is not hard to verify that, for any intervals I and J, $\{i + j \mid i \in I, j \in J\} = I \oplus J$, and the same holds for subtraction, multiplication, and division. However, the distributive law does not hold for intervals – it only holds that $A \odot (B \oplus C) \subset (A \odot B) \oplus (A \odot C)$. The reason is that when evaluating the right-hand side of the expression, both occurrences of A are treated separately, which means that the interval arithmetic does not capture the dependency between sub-expressions. Therefore, we have to specify an evaluation method for polynomial evaluation at an interval. For $f = \sum_{i=0}^{n} a_i t^i$, we define

$$\Box f(I) = a_0 + I(a_1 + I(a_2 + \ldots (a_{n-1} + I a_n) \cdots).$$

This evaluation method is also called the *Horner scheme*. The image of f at I is contained in $\Box f(I)$, although the interval arithmetic might considerably overestimate the image. Moreover, if we consider a sequence of intervals $I_0 \supset I_1 \supset \ldots$ with widths converging to zero, then the width of the sequence $\Box f(I_0) \supset \Box f(I_1) \supset \ldots$ converges to zero as well. The next lemma quantifies this convergence for the case of an integer polynomial f.

Lemma 2.5.20. *Let $f = \sum_{i=0}^{n} a_i t^i$ of magnitude (n, τ), and I be an interval of width $0 < \varepsilon < 2$. Then, for each $\alpha \in I$ and each $y \in \Box f(I)$, we have*

$$|y - f(\alpha)| \leq 2^n \varepsilon 2^\tau \max\{1, |\alpha|\}^{n-1}.$$

Proof. Fix some $y \in \Box f(I)$. There exist values $\nu_1, \ldots, \nu_n \in [-\frac{\varepsilon}{2}, \frac{\varepsilon}{2}]$ such that

$$y = a_0 + (\alpha + \nu_1) \cdot (a_1 + (\alpha + \nu_2) \cdot (a_2 + \ldots) + (\alpha + \nu_n) \cdot a_n) \ldots)$$

Through induction on n, for $n = 1$, we have $|y - f(\alpha)| = \nu_1 a_1 \leq \varepsilon 2^\tau$, and the claim is satisfied. For $n > 1$, we write

$$f(\alpha) = a_0 + \alpha \cdot \tilde{f}(\alpha), \quad y = a_0 + (\alpha + \nu_1)\tilde{y}$$

with $\tilde{f} = \sum_{i=0}^{n-1} a_{i+1} t^i$ and $\tilde{y} \in \Box \tilde{f}(I)$. So, we now have

$$|y - f(\alpha)| = |\nu_1 \tilde{y} + \alpha(\tilde{y} - \tilde{f}(\alpha))|.$$

Since

$$\begin{aligned} |\tilde{y}| &\leq |a_1| + (|\alpha| + |\nu_2|) \cdot (|a_2| + \ldots) + (|\alpha| + |\nu_n|) \cdot |a_n|) \ldots) \\ &\leq 2^\tau \sum_{i=0}^{n-1} (|\alpha| + \frac{\varepsilon}{2})^i \\ &\leq 2^\tau \sum_{i=0}^{n-1} (2 \max\{|\alpha|, 1\})^i \\ &\leq 2^\tau 2^n \max\{|\alpha|, 1\}^{n-1}, \end{aligned}$$

it follows together with the induction hypothesis that

$$\begin{aligned} |y - f(\alpha)| &= |\nu_1 \tilde{y} + \alpha(\tilde{y} - \tilde{f}(\alpha))| \\ &\leq \frac{\varepsilon}{2} 2^\tau 2^n \max\{|\alpha|, 1\}^{n-1} + \alpha \left(2^n \frac{\varepsilon}{2} 2^\tau \max\{1, |\alpha|\}^{n-2} \right) \\ &\leq \frac{\varepsilon}{2} 2^\tau \max\{|\alpha|, 1\}^{n-1} (2^n + 2^n) \\ &\leq \frac{\varepsilon}{2} 2^\tau \max\{|\alpha|, 1\}^{n-1} 2^{n+1}. \quad \Box \end{aligned}$$

Algorithm 2.17. Approximating $f(\alpha)$

INPUT: $\delta > 0$, $f \in \mathbb{Z}[t]$, and $\alpha = (g, I)$
OUTPUT: An interval J containing $f(\alpha)$ with $\mathrm{w}(J) < \delta$

```
procedure APPROX_EVAL(f, (g, I), δ)
    J ← □f(I)
    while w(J) ≥ δ do
        Refine I using QIR
        J ← □f(I)
    end while
    return J
end procedure
```

We turn to the complexity of interval arithmetic. Assume that the bitsize of all interval boundaries is bounded by σ. The basic arithmetic interval operations yield the same bit complexity as the same operations on integers of bitsize σ, up to a constant factor. The same is true when evaluating a polynomial at an interval.

With these basic properties of interval arithmetic in mind, we can look at further algorithms for algebraic numbers. Consider the following problem: Given $\delta > 0$, $f \in \mathbb{Z}[t]$, and some algebraic number $\alpha \in \mathbb{R}$ in isolating interval representation (g, I), the goal is to compute an interval J containing $f(\alpha)$, such that $w(J) \leq \delta$.

The solution is absolutely straightforward, following the techniques just dicussed: compute $J := \Box f(I)$. If its width is greater than δ, refine I and start over, otherwise, return J (Algorithm 2.17). The complexity statement for Algorithm 2.17, although not hard to derive, appears in a quite complicated form because we do not assume here that f and g are of the same magnitude. In our later application, the magnitude of g will be larger than that of f, and it will yield more precise bounds if these parameters are distinguished. Furthermore, we analyze the case that Algorithm 2.17 is applied for each root of g, since the same bound is obtained in this situation as for a single root.

Theorem 2.5.21. *Let f be of magnitude (n, τ) and g of magnitude (n', τ') with $n \leq n'$ and $\tau \leq \tau'$. Set $L_\delta = \log \frac{1}{\delta}$. Then, Algorithm 2.17, applied to all real roots of g runs in a total bit complexity of*

$$\tilde{O}(n'^4 \tau'^2 + n'^3 (L_\delta + \tau' n)).$$

Proof. Let I be the isolating interval of some real root α of g. Note first that if I is refined to size

$$\varepsilon := 2 \frac{\delta}{2^{n+1} 2^\tau \max\{1, |\alpha|\}^{n-1}},$$

the distance of $y \in \Box f(I)$ to $f(\alpha)$ is bounded by

$$|y - f(\alpha)| \leq \frac{1}{2}\delta$$

using Lemma 2.5.20, and by the triangle inequality, $\mathrm{w}(\Box f(I)) \leq \delta$ holds.

Thus, I must be refined to precision ε. Note that

$$L_\varepsilon = \tilde{O}(L_\delta + n + \tau + n\tau') = \tilde{O}(L_\delta + n\tau')$$

($|\alpha| \in O(\tau')$, because it is a root of g). Refining all I's to size ε takes

$$\begin{aligned} & \tilde{O}(n'^4\tau'^2 + n'^3 L_\varepsilon) \\ = \; & \tilde{O}(n'^4\tau'^2 + n'^3(L_\delta + n\tau')) \end{aligned}$$

bit operations by Theorem 2.5.16 (strong root isolation), which is the desired bound.

We have to argue why the evaluations $J \leftarrow \Box f(I)$ do not increase the complexity. For that, we have to recall the internals of the quadratic interval refinement method. In each step, g is evaluated at the interval boundaries of I, so each such step is at least as costly as such an evaluation. We have argued above that an interval evaluation is as costly as the evaluation of one of its boundaries. Since f is of smaller magnitude than g, it follows that this evaluation is not more costly than the preceding qir call. There only remains the initial interval evaluation to perform, but this step is dominated by the overall complexity. □

We further generalize the result just obtained. Assume that we do not just approximate one polynomial f at the roots of g, but a whole sequence $f_1, \ldots, f_k$. In this case, the bound from the theorem above could just be multiplied by k to obtain a lower bound. It is not surprising that we can do better, since the bottleneck in the above analysis is to approximate the α_i to sufficient precision. Clearly, this is not done k times from scratch when considering several polynomials.

Theorem 2.5.22. *Let $f_1, \ldots, f_k$ be of magnitude (n, τ) and g of magnitude (n', τ') with $n \le n'$ and $\tau \le \tau'$. Set $L_\delta = \log \frac{1}{\delta}$. Then, Algorithm 2.17, applied to all real roots of g, runs in total bit complexity*

$$\tilde{O}(n'^4\tau'^2 + n'^3(L_\delta + \tau' n) + kn'^2 n(L_\delta + \tau' n)).$$

Proof. The previous proof shows that, once α is refined to precision ε, $\mathrm{w}(\Box f(\alpha)) \le \delta$ for any f of magnitude (n, τ). Thus, we still need not more than $\tilde{O}(n'^4\tau'^2 + n'^3(L_\delta + n\tau'))$ bit operations for the refinements. The additional summand $kn'^2 n(L_\delta + n\tau')$ arises because one initially has to compute kn'^2 interval evaluations, and the bitsizes of the interval boundaries are bounded by $L_\varepsilon = \tilde{O}(L_\delta + n\tau')$. □

We next discuss a related problem: Given $f \in \mathbb{Z}[t]$, and $\alpha \in \mathbb{R}$ as before, evaluate the sign of $f(\alpha)$. The solution is similar to the comparison of algebraic numbers. First, we check whether α is a root of f by considering the gcd of f and g (where g is the defining polynomial of α). If α is not a root, we simply refine its isolating interval until $0 \notin \Box f(I)$ (Algorithm 2.18).

In the complexity analysis, we again bound the time needed to compute the signs of $f(\alpha)$ for all real roots of g. For this application, we assume for simplicity that f and g are of the same magnitude (since this will also be the case in our applications).

Theorem 2.5.23. *Let f and g be of magnitude (n, τ). Computing the sign of $f(\alpha)$ for each $\alpha \in V(g)$, using Algorithm 2.18, has a bit complexity of*

$$\tilde{O}(n^4(n + \tau^2)).$$

Proof. The first part of Algorithm 2.18, concerning the gcd and its evaluation has a bit complexity of $\tilde{O}(n^3\tau)$ per call by the same analysis as for Algorithm 2.15 (Equal). Doing this for all (up to n) real roots of g requires $\tilde{O}(n^4\tau)$ bit operations in total.

Algorithm 2.18. Sign of $f(\alpha)$

INPUT: $f \in \mathbb{Z}[t]$, and $\alpha = (g, I)$
OUTPUT: The sign of $f(\alpha)$ (1,-1, or 0)

procedure SIGN$(f, (g, I))$
  $h \leftarrow \gcd(f, g)$
  if $I = [c, c]$ and $h(c) = 0$ return 0
  if $I = [c, d]$ and $h(c)h(d) < 0$ return 0 ▷ Otherwise, $f(\alpha) \neq 0$
  while $0 \in \square f(I)$ **do**
    Refine I using QIR
  end while
  return sign(I) ▷ sign(I) = the sign of any value in I
end procedure

It remains to bound the cost of the refinement loop for each α. As before, we can concentrate on the qir calls, because we charge the cost for an interval evaluation to the cost of the preceding qir call.

Fix some root α of g with $f(\alpha) \neq 0$, and let I_α be its isolating interval. Define

$$\delta_\alpha := \frac{|f(\alpha)|}{2^{n+1}2^\tau \max\{1, |\alpha|\}^n}.$$

A simple estimation shows that $\delta_\alpha < 1$ for all $\alpha \in \mathbb{C}$. If $\mathrm{w}(I_\alpha) < \delta_\alpha$, it follows from Lemma 2.5.20 that for all $y \in I_\alpha$

$$|y - f(\alpha)| \leq \frac{|f(\alpha)|}{2\max\{1, |\alpha|\}} \leq \frac{|f(\alpha)|}{2},$$

and thus $0 \notin I_\alpha$.

The remainder of the proof works as follows. We will show below that

$$\sum_{\alpha \in V_\mathbb{R}(g) \setminus V_\mathbb{R}(f)} \log \frac{1}{\delta_\alpha} = O(n(n+\tau)).$$

It follows that $\log \frac{1}{\delta_\alpha} = O(n(n+\tau))$ for every single root as well. Consequently, there is some width ε with $L_\varepsilon = O(n(n+\tau))$ that is sufficient for each α, and refining the isolating intervals to size ε requires

$$\tilde{O}(n^4\tau^2 + n^3(n(n+\tau))) = \tilde{O}(n^4(\tau^2+n))$$

bit operations, with Theorem 2.5.16, which proves our claim.

It is left to show that

$$\sum_{\alpha \in V_\mathbb{R}(g) \setminus V_\mathbb{R}(f)} \log \frac{1}{\delta_\alpha} = O(n(n+\tau)).$$

Note first that instead of talking about all roots of g that are not roots of f, we can also talk about the roots of $\tilde{g} := \frac{g}{\gcd(f,g)} \in \mathbb{Z}[t]$. The proof is almost analogous to that of

Theorem 2.5.8: Since $\delta_\alpha \leq 1$ for all $\alpha \in \mathbb{C}$, we can replace $V_{\mathbb{R}}(\tilde{g})$ by $V_{\mathbb{C}}(\tilde{g})$ in the sum, and we obtain

$$\begin{aligned}\sum_{\alpha\in V_{\mathbb{R}}(\tilde{g})} \log\frac{1}{\delta_\alpha} &\leq \sum_{\alpha\in V_{\mathbb{C}}(\tilde{g})} \log\frac{1}{m_\alpha} \\ &\leq \sum_{\alpha\in V_{\mathbb{C}}(\tilde{g})} \log\frac{2^{n+2}2^\tau \max\{1,|\alpha|\}^n}{|f(\alpha)|} \\ &\leq n(n+2)+n\tau+n\log\prod_{\alpha\in V_{\mathbb{C}}(\tilde{g}} \max\{1,|\alpha|\} + \log\prod_{\alpha\in V_{\mathbb{C}}(\tilde{g}} \frac{1}{|f(\alpha)|} \\ &= n(n+2)+n\tau+n\log\frac{\mathrm{Mea}(\tilde{g})}{\mathrm{lcf}(\tilde{g})} + \log\frac{\mathrm{lcf}(f)^n}{\mathrm{res}(f,\tilde{g})}.\end{aligned}$$

In the last step, we have used Theorem 2.3.6. Noting that $\mathrm{lcf}(\tilde{g})$ and $\mathrm{res}(f,\tilde{g}) \in \mathbb{Z}$, this further simplifies to

$$\leq \quad n(n+2)+n\tau+n\log\mathrm{Mea}(\tilde{g})+n\log\mathrm{lcf}(f).$$

Since $\tilde{g}$ is of magnitude $(n, n+\tau)$, $\log\mathrm{Mea}(\tilde{g}) = O(n+\tau)$, and $\log\mathrm{lcf}(f) = O(\tau)$. Thus, the sum is bounded by $O(n(n+\tau))$ as required. □

Again, we generalize the analysis to the case of more than one polynomial. Assume that a sequence of polynomials $f_1,\ldots,f_k$, all of magnitude (n,τ), is evaluated at the real roots $\alpha_1,\ldots,\alpha_s$ of another polynomial. A complexity of $\tilde{O}(kn^4(n+\tau^2))$ is immediately obvious, but we can do better.

Theorem 2.5.24. *Let $f_1,\ldots,f_k$ and g be of magnitude (n,τ). Computing the signs of $f_1(\alpha),\ldots,f_k(\alpha)$ for each $\alpha \in V(g)$ using Algorithm 2.18 has a bit complexity of*

$$\tilde{O}(kn^4(n+\tau)+n^4(n+\tau^2)).$$

Proof. Computing the k gcd's of g with the f_i's is in $\tilde{O}(kn^3\tau)$, so this step is covered by the bit complexity. The proof of the preceding theorem has shown that, when each isolating interval is refined to a fixed ε with $L_\varepsilon = O(n(n+\tau))$, the sign of $f(\alpha)$ can be determined for any f of magnitude (n,τ). Thus, the bitsize of the intervals in the algorithm is bounded by $O(n(n+\tau))$.

For each pair of the up to kn many pairs (f_i,α_j), we have to perform two initial evaluations (to check for zero), plus one initial interval arithmetic evaluation (to obtain $\Box f(I)$). Because the interval boundaries are in $O(n(n+\tau))$, these steps are in $O(kn\cdot nM(n^2(n+\tau))) = \tilde{O}(kn^4(n+\tau))$.

The cost of the qir loop is bounded by $\tilde{O}(n^4(\tau^2+n))$ as before, since the additional interval arithmetic steps can be charged to the qir calls. Summing up both quantities yields the result. □

Summary

Isolating intervals, as returned by the Descartes method, allow the representation of arbitrary algebraic numbers, and processing such numbers in an efficient way is possible. We have seen how to approximate algebraic numbers using the quadratic interval refinement method. Based on this, we have described adaptive algorithms to, for instance, compare algebraic numbers, evaluate the sign of an algebraic number α, or approximate $g(\alpha)$. In all cases, considering all roots associated with α (instead of concentrating on a single root) allowed us to derive better worst-case bounds for the computation.

2.6. Computation with polynomials with bitstream coefficients

So far, we have mainly assumed the input polynomials of our algorithms to have integer coefficients. The Descartes method (Section 2.4.4) is not in principle restricted to integer coefficients, but can be applied to any polynomial with real number coefficients as long as the arithmetic operations can be performed in an exact way (Algorithm 2.9). However, the cost of such exact arithmetic can render the practical root isolation infeasible, for instance, when coefficients are arbitrary algebraic numbers, or when dealing with very big integer coefficients. For polynomials with transcendental coefficients, it is not even known how the required operations can be performed in an exact manner.

To overcome these problems, a variant of the Descartes method has been proposed that works only by approximating the polynomial's coefficients by floating-point numbers. More precisely, for any polynomial coefficient $a \in \mathbb{R}$ and any precision $p \in \mathbb{Z}$, an integer m must be returned such that $|m - a2^p| \leq 1$. If this operation is supported for the coefficients, we call them to be in bitstream representation [Eig08, Def 3.35], which captures the intuition that arbitrary many bits of the coefficients can be obtained, but one can never decide whether all remaining bits are zero. We emphasize, however, that the name "bitstream" does not imply that the returned approximation coincides with the first p bits of the exact binary representation of the real number. To give an example, if $a = 0.01101$ (in binary representation), it is legal that the bitstreams 1, 10, 011, 0110 are returned for $p = 1, 2, 3, 4$, respectively.

The variant of the Descartes method that works with coefficients in bitstream representation is called the *bitstream-Descartes* method. A bitstream constitutes a slight perturbation of the coefficient, and thus, by bitstreaming all coefficients, one obtains a slightly perturbed polynomial $\tilde{f}$. The idea is that the isolating intervals for $\tilde{f}$ remain isolating for f (possibly with a small extension), since the roots of a polynomial depend continuously on its coefficients.

Despite this understandable argument, several difficulties arise when designing an efficient and reliable root isolator. Most of them boil down to the (exact) zero test that is simply unavailable when working with coefficient approximations. That means, for instance, that Step 10 of Algorithm 2.9 (where $f(m) = 0$ is checked for the bisection midpoint of an interval) cannot be decided just by looking at approximations, regardless of the precision. Another challenge for the algorithm is to adaptively choose a sufficient working precision, since an a priori worst-case bound usually overestimates the needed precision considerably. Still, such a bound will be necessary for a complexity analysis.

Two different approaches have been presented recently. The first one is mainly worked out by Eigenwillig [Eig08, EKK$^+$05]. Its idea is to conceptually consider all polynomials within a given approximation precision, explicitly coping with the uncertainty in the number of sign variations of f. He can show that the computation path of the exact polynomial can be followed, assuming that the polynomial is sufficiently large at its subdivision points. A randomized choice of the subdivision points (instead of always cutting in the middle as in the usual Descartes method) ensures this property with high probability, if the precision was high enough.

A deterministic variant is presented by Mehlhorn and Sagraloff [MS09]. Therein, one chooses a concrete approximation $\tilde{f}$ of f (with rational coefficients) and applies a slight variant of Algorithm 2.9. Upon succeeding, the isolating intervals of $\tilde{f}$ are slightly enlarged to both sides, such that the extended interval is also isolating for f. If the intervals for $\tilde{f}$ become too small during the Descartes method, the precision is increased (the meaning of "small" depends on the chosen precision).

In the following, we will not just apply the bitstream-Descartes version to polynomials with algebraic number coefficients but we will additionally discuss variants of the algorithm that work for non-square-free polynomials, using additional knowledge of the input polynomial (Section 2.6.2). To be able to describe those variants, we need to describe one of the bitstream variants in more detail. We have decided to use the variant by Mehlhorn and Sagraloff for our theoretical description for the following reasons.

- Their variant seems conceptually simpler since it always talks about one particular approximation of the polynomial instead of considering all possible approximation at the same time.
- It can be formulated completely in power basis, whereas Eigenwillig's version needs an initial conversion into the Bernstein basis. Although this conversion is not too complicated, from neither a theoretical nor a practical point of view, it still requires a substantial body of additional concepts.
- One of the variants for polynomials with multiple roots that will be discussed, the *m-k-bitstream-Descartes* variant, has already been analyzed completely in [MS09] in the context of the deterministic variant. In [Eig08], a similar analysis has been started but is not complete. Nertheless, some of the results therein are of great significance also within this thesis and will be cited at the appropriate position.

Despite these arguments in favor of the deterministic approach, the randomized algorithm also has an advantage: a complete implementation has been provided by Eigenwillig. Such an implementation for the deterministic Descartes method is missing. Thus, we are in the somewhat uncomfortable situation in which we use one variant for theoretical considerations, but the other for the implementation. Still, we believe that the complexity bounds derived for the deterministic version also remain valid for the randomized approach (this has been proven for the square-free version at least), and in turn, that an implementation of the deterministic algorithm should show a similar performance compared to Eigenwillig's implementation (first experiments with a prototypical deterministic isolator back up this guess).

2.6.1. The bitstream-Descartes method for square-free polynomials

We (briefly) discuss the *Deterministic bitstream-Descartes method*. The results of this section are a summary of [MS09]. As we have mentioned, the intuition behind the method is that the roots of a polynomial continuously depend on its coefficients. The next theorem, attributed to Schönhage [Sch85], gives a quantified version of this statement (we state a slightly weaker form adapted to our situation).

Theorem 2.6.1 (Schönhage). *Let f be a polynomial of degree n with roots $\alpha_1, \ldots, \alpha_n \in \mathbb{C}$ such that $|\alpha_i| < \frac{3}{4}$. Let $\mu \leq 2^{-7n}$ and f^* be a polynomial of degree n with complex roots*

$\alpha_1^*, \ldots, \alpha_n^*$ such that

$$\|f - f^*\|_1 < \mu \|f\|_1.$$

Then, up to a permutation of the α_i^*'s, it holds that

$$|\alpha_i - \alpha_i^*| < 9\sqrt[n]{\mu}.$$

In particular, all roots of f^* are in the complex unit circle.

For a given f of degree n, fix some $\mu < 2^{-7n}$, and consider any approximation polynomial f^* as in Theorem 2.6.1. The principal idea is to apply the Descartes method on f^*, and to extend the isolating intervals to each side by $9\sqrt[n]{\mu}$ to obtain isolating intervals for f, but several problem arise here:

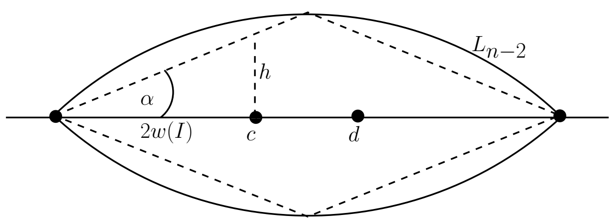
- Real roots of f can turn into non-real roots of f^*, and vice versa.
- The extension of the isolating intervals might lead to an overlap of isolating intervals.
- The approximation f^* is not always square-free, and the Descartes method might diverge.

To avoid real roots becoming non-real (and vice versa) when going from f^* to f, we define something like a "safety margin": whenever the subdivision stops for some interval I, we must ensure that a stripe of the form $I \times [-ih, ih]$ (with i being the complex unit, and $h > 0$) is free of non-real roots. To avoid overlapping intervals, we must ensure that isolating intervals have a distance of at least $18\sqrt[n]{\mu}$. The third problem will be solved by the final algorithm more or less automatically, thus we skip its discussion for now.

We introduce a slightly adapted Descartes test that helps to overcome the first two problems. For any interval $I = (c, d)$, define the extended interval $I^+ := (c - 2(d - c), d + 2(d - c))$. I^+ is five times as big as I, and has I at its center. Moreover, we define $v := \mathrm{Var}(f^*, I)$ as in Section 2.4.4, and $v^+ := \mathrm{Var}(f^*, I^+)$. Clearly, $v^+ \geq v$.

Lemma 2.6.2. *If $v^+ \leq 1$, there is no non-real root in the region $I \times [-2i\frac{w(I)}{n}, 2i\frac{w(I)}{n}]$.*

Proof. Since $v^+ \leq 1$, we can use the contraposition of the first part of Theorem 2.4.30 (Obreshkoff's Theorem) It follows that the Obreshkoff lens L_{n-2} contains less than two roots, and thus no non-real roots. Consider the following picture.



Note that $\alpha = \frac{\pi}{2n}$ follows by the inscribed angle theorem (see the remark after Definition 2.4.29). We can thus bound

$$h = 2w(I)\tan(\frac{\pi}{2n}) \geq \frac{2w(I)}{n}$$

(using $\tan(c\frac{\pi}{2}) \geq c$ for $0 < c < 1$). $\square$

Algorithm 2.19. The Descartes method with expanded intervals

INPUT: $f^* \in \mathbb{R}[t]$ with all roots in the complex unit circle
OUTPUT: Isolating intervals for all real roots of f^*

procedure DESCARTES_EXTENDED(f^*)
  $Q \leftarrow \{(0,1)\}$
  while Q is not empty **do**
    Let $J = (c,d)$ be the first element of Q. Remove J from Q.
    $v^+ \leftarrow \mathrm{Var}(f^*, J^+)$ ▷ $J^+ = (c - 2(d-c), d + 2(d-c))$
    if $v^+ > 1$ **then**
      $m \leftarrow \frac{c+d}{2}$
      if $f(m) = 0$ **then** append $[m,m]$ to the output list
      end if
      Append (c,m) and (m,d) to Q
    else
      $v \leftarrow \mathrm{Var}(f^*, J)$
      if $v = 0$ **then** do nothing
      end if
      if $v = 1$ **then** append I to the output list
      end if
    end if
  end while
end procedure

Algorithm 2.19 defines the new version of the Descartes method. Intervals are always subdivided as long as $v^+ > 1$. Once $v^+ \leq 1$, the interval is removed from the queue, and in the case where $v = 1$, it is added to the output list.

It is not hard to see that Algorithm 2.19 terminates if f^* is square-free, because the one-circle and two-circle theorems also apply eventually for the extended intervals I^+. If the intervals do not become too small during the subdivision, we can even deduce the isolating intervals of f.

Theorem 2.6.3. *Let f, μ, f^* be chosen according to Theorem 2.6.1. Assume that Algorithm 2.19 returns the isolating intervals $I_1, \ldots, I_m$, with $I_j = (c_j, d_j)$ (or $I_j = [c_j, d_j]$ with $c_j = d_j$) Also, assume that no interval of size less than $9n\sqrt[n]{\mu}$ is produced during the subdivision. Then, f has the same number of real roots as f^*, and the intervals $I'_1, \ldots, I'_m$ with $I'_j = (c_j - 9\sqrt[n]{\mu}, d_j + 9\sqrt[n]{\mu})$ are isolating intervals for f.*

Proof. First, we argue that any pair of isolating intervals of f^* has at least a distance of $18\sqrt[n]{\mu}$. For two roots α and β with $\alpha < \beta$, let I be the interval that contains α, and let J be the one that contains β (if I is a singleton interval, choose instead the interval that has α as its left boundary for I). W. l. o. g., $w(I) \geq w(J)$. Since $\mathrm{Var}(f^*, I^+) = 1$, $\beta \notin I^+$, and J is disjoint from I^+. Therefore, the distance between I and J (i.e., the minimal distance of an element in I to an element in J) is at least $2w(I) \geq 18\sqrt[n]{\mu}$. The same argument also shows that distance between any pair of real roots is more than $18\sqrt[n]{\mu}$.

The above argument shows that the extended intervals $I'_1, \ldots, I'_m$ are disjoint. Using Lemma 2.6.2 on each interval with $\mathrm{Var}(f^*, I^+) \leq 1$, we see that each complex root of f^*

has a imaginary part of more than $2\frac{w(I)}{n} > 2\frac{9n\sqrt[n]{\mu}}{n} = 18\sqrt[n]{\mu}$. That implies that no non-real root of f^* can correspond to a real root of f. Also, vice versa, a real root of f^* cannot transform into a non-real root of f. Assuming some root α of f^* does so, another root of f^* would turn into its complex conjugate. Since α has a distance of at least $18\sqrt[n]{\mu}$ to all other roots (real or non-real), and the roots cannot move more than $9\sqrt[n]{\mu}$ according to Theorem 2.6.1, this is impossible. This shows that each I'_j contains precisely one real root of f. □

If we can ensure that Algorithm 2.19 does not produce an interval of length smaller than $9n\sqrt[n]{\mu}$, the roots of f can be isolated just by looking at its approximation f^*. Not surprisingly, this condition will be satisfied for a small enough μ. To quantify a sufficient value of μ, we define, for a polynomial $g = \prod_{i=0}^{m}(t-\alpha_i)^{e_i}$,

$$s(g) := \min\{\min\{|\alpha_i - \alpha_j| \mid \alpha_i, \alpha_j \in \mathbb{R}\}, \min\{\mathrm{Im}(\alpha_i) \mid \alpha_i \notin \mathbb{R}\}\},$$

where $\mathrm{Im}(c)$ denotes the imaginary part of a complex number. Note the small difference compared to the related $\mathrm{sep}(f)$ (Definition 2.4.32); the distance between complex roots is not taken into account in $s(g)$. If f, μ, and f^* are chosen as in Theorem 2.6.1 (Schönhage's Theorem), it directly follows that $|s(f) - s(f^*)| < 18\sqrt[n]{\mu}$. Moreover, Algorithm 2.19 applied to g does not ever subdivide an interval I of size $\frac{s(g)}{5}$, because for such intervals, $w(I^+) \le s(g)$ and thus, $\mathrm{Var}(g, I^+) \in \{0,1\}$ (because either the one-circle or the two-circle theorem applies).

Lemma 2.6.4. *Let f, μ and f^* be as before. If*

$$\mu \le \min\left\{2^{-7n}, \left(\frac{s(f)}{63n}\right)^n\right\},$$

Algorithm 2.19 applied on f^* does not produce an interval of size smaller than $9n\sqrt[n]{\mu}$.

Proof. The condition $\mu \le \left(\frac{s(f)}{63n}\right)^n$ is equivalent to $s(f) \ge 63n\sqrt[n]{\mu}$. Thus we can estimate

$$s(f^*) \ge s(f) - 18\sqrt[n]{\mu} \ge 63n\sqrt[n]{\mu} - 18\sqrt[n]{\mu} \ge 45n\sqrt[n]{\mu}.$$

Since no interval of size smaller than $\frac{s(f^*)}{5}$ is subdivided, this implies that no interval becomes smaller than $9n\sqrt[n]{\mu}$. □

We formulate the complete bitstream-Descartes method next. So far, our considerations have only applied to polynomials with roots in the unit circle, but this is simply a matter of scaling: by an initial transformation $t \mapsto 4B(t-\frac{1}{2})$, where B is a root bound according to Theorem 2.2.11, all roots are inside a complex disc centered at $(\frac{1}{2}, 0)$ of radius $\frac{1}{4}$. The transformed polynomial has bitstream coefficients as well.

We set $\mu := 2^{-7n}$. For μ, we compute an approximation f^* of f as follows. For a coefficient a_i of f, we compute an approximation a_i^* of f^* with $|a_i - a_i^*| \le \frac{\mu\|f\|_1}{n+2}$. The f^* thus obtained has the property that

$$\|f - f^*\|_1 \le (n+1)\|f - f^*\|_\infty \le (n+1)\frac{\mu\|f\|_1}{n+2} < \mu\|f\|_1,$$

so f^* is a polynomial with the properties of Theorem 2.6.1 (Schönhage's Theorem). Algorithm 2.19 is applied to f^*, but it is interrupted as soon as an interval of size smaller

Algorithm 2.20. The (deterministic) bitstream-Descartes method

INPUT: $f \in \mathbb{R}[t]$ OUTPUT: Isolating intervals for all real roots of f

procedure BITSTREAM_DESCARTES(f)
  $\overline{f} \leftarrow f(4B(t-\frac{1}{2}))$
  $\mu \leftarrow 2^{-7n}$
  while true **do**
    Approximate each coefficient of $\overline{f}$ by $\frac{\mu\|p\|_1}{n+2} \rightsquigarrow f^*$
    DESCARTES_EXTENDED(f^*)
    if interval of size smaller than $9n\sqrt[n]{\mu}$ is produced **then** $\mu \leftarrow \mu^2$
    else return $4B(I'_1-\frac{1}{2}),\ldots,4B(I'_m-\frac{1}{2})$ with $I'_1,\ldots,I'_m$ as defined in Theorem 2.6.3
    end if
  end while
end procedure

than $9n\sqrt[n]{\mu}$ is produced, and μ is squared in this case. If no such interval is produced, the returned isolating intervals are extended on both sides by $9\sqrt[n]{\mu}$ to obtain isolating intervals for f. See Algorithm 2.20 for pseudo-code.[18]

Note that an unfortunate choice of f^* might yield the application of Algorithm 2.19 to a non-square-free polynomial, which leads to divergence. Therefore, it is important that the execution of Algorithm 2.19 be interrupted as soon as an interval of size less than $9n\sqrt[n]{\mu}$ is detected; it is not enough to wait for its termination and check the isolating intervals afterwards.

The complexity analysis works in a way similar to that of the integer Descartes method, by bounding the subdivision tree of the method, and exploiting Lemma 2.6.4 to bound the bitsize of the approximation polynomial. We state the bound established in [MS09], assuming asymptotically fast Taylor shifts

Theorem 2.6.5. *Let $f = \sum a_i t^i \in \mathbb{R}[t]$ be a square-free polynomial with $|\mathrm{lcf}(f)| \geq 1$ and $|a_i| < 2^{\tau-1}$ for all i. The bit complexity of Algorithm 2.20 is*

$$\tilde{O}\left(n^3\left(\tau + \log\frac{1}{\mathrm{sep}(f)}\right)^2\right).$$

The coefficients of f need to be approximated with $O(n(\log n + \tau + \log\frac{1}{\mathrm{sep}(f)}))$ bits after the binary point.

Compared to [MS09], we also replaced $s(f)$ by $\mathrm{sep}(f)$ in Theorem 2.6.5. This is done for later convenience, but requires a justification.

Lemma 2.6.6. *For a square-free polynomial f, $\mathrm{sep}(f) \leq 2s(f)$*

Proof. Assume that $\mathrm{sep}(f) > 2s(f)$. This implies that there is some non-real root α with an imaginary part smaller than $\frac{\mathrm{sep}(f)}{2}$. This is already a contradiction, since its distance to the complex conjugate (which is also a root) is smaller than $\mathrm{sep}(f)$. ☐

[18] Of course $\|f\|_1$ cannot be known exactly and must be bounded from above in the algorithm

2.6.2. Variants of the bitstream-Descartes method

A crucial prerequisite of the Descartes and bitstream-Descartes methods was the square-freeness of the input polynomials. The termination condition for the Descartes method does not apply anymore for a multiple real root. Also, applying the bitstream-Descartes method appears hopeless, since a double root might turn into a pair of non-real roots, no matter how closely we approximate the exact polynomial.

Still, we will frequently face non-square-free polynomials and we want to treat them using the bitstream-Descartes method. To give a motivation, consider an algebraic curve $V(f)$ with a defining bivariate integer polynomial f. We have discussed critical x-coordinates of $V(f)$, which are algebraic numbers in general. Let α be such a value. A crucial operation in our arrangement algorithm will be to compute the fiber of f at α. The polynomial $f(\alpha, y)$ has algebraic coefficients; we wish to apply the bitstream-Descartes method to it instead of the usual Descartes method to avoid symbolic computations. However, $f(\alpha, y)$ has multiple roots since α is a critical x-coordinate.

What can be done for non-square-free polynomials? One possible way out is to compute its square-free part initially, and apply Algorithm 2.20 to it. But this has several disadvantages. First of all, computing the square-free part can be a costly operation for polynomials with algebraic coefficients, and the coefficients of the square-free part might be considerably more complicated. Another issue is that, depending on the coefficient type of the polynomial, computing a gcd, and thus the square-free part, might be impossible.

We take a different approach. The overall idea is that some additional information about the polynomial is accessible from a different source. This information usually arises out of symbolic computations, but they are less costly than computing the square-free part. Then, the bitstream-Descartes method is applied to the (non-square-free) polynomial, but with a modified termination condition. We describe two variants of this approach. Both might appear somewhat artificial upon first reading – their design is strongly coupled to our algorithm for the analysis of algebraic curves, which follows in Section 3.2. Still, we decided to discuss them already in this chapter, because they are, after all, root isolation algorithms, and might be useful in other applications as well.

m-k-bitstream-Descartes

Given $f \in \mathbb{R}[t]$, not necessarily square-free, assume that we additionally know the quantities:

$$\begin{aligned} m &:= \#\{\alpha \in \mathbb{R} \mid f(\alpha) = 0\} \\ k &:= \deg \gcd(f, f'). \end{aligned}$$

We describe a method, called *m-k-bitstream-Descartes method* to isolate the real roots of f. As with the usual bitstream-Descartes method, we will only use coefficient approximations of f. There is no global success guaranteed for the algorithm – it might fail in certain situations. However, we will prove that it always terminates and, if f has at most one multiple root over $\mathbb{C}$, the algorithm never fails, which means that the isolating intervals have been computed.

The algorithm was first described in the author's Master's thesis [Ker06], and has been used to compute the geometric-topological analysis of curves [EKW07] and surfaces [BKS08]. In [MS09], the deterministic version of it has been discussed, and a complexity analysis has been provided.

For a simplified introduction, we assume for the moment that the input polynomial f is given by exact integer coefficients so that the exact sign variation of an interval can be obtained. In this case, the algorithm is relatively straightforward to formulate. Apply the Descartes method (Algorithm 2.9) on the non-square-free f. If, during the execution, there are already $m-1$ intervals in the output list (i.e., intervals with sign variation 1) and the queue of active intervals Q contains only one remaining interval, then this last interval is appended to the output list, and the method stops by returning the output list (we call this termination condition the *m-case*, or *success condition*). If, during the subdivision, all intervals in the queue have a sign variation of less than $k+1$, the method is interrupted and returns a failure indicator (we call this the *k-case* or *failure condition*).

This method always terminates: Because of Theorem 2.4.30 (Obreshkoff's Theoerem), each root of f is eventually contained in an interval whose sign variation equals the multiplicity of the root. Assume that the subdivision is in this state. We distinguish two cases. If f has at most one multiple root over $\mathbb{R}$, the success condition is clearly satisfied, and the algorithm terminates. If f has more multiple roots, each of them has a multiplicity of less than $k+1$ because each multiple root α contributes $\operatorname{mult}(\alpha, f) - 1 \geq 1$ to $k = \deg\gcd(f, f')$. Thus, in this case, the failure condition is satisfied. Note that in this argument, we heavily rely on the fact that the subdivision tree is explored in a BFS-like manner; using a DFS strategy would lead to infinite refinement in the presence of multiple roots.

Observe that if f has exactly one multiple root α over $\mathbb{C}$, the failure condition is never satisfied, because in this case $\alpha \in \mathbb{R}$, and $\operatorname{mult}(\alpha, f) = k+1$, thus there is always one interval that counts $k+1$.[19] It follows that, indeed, the method is always successful for polynomials with only one multiple complex root.

What are the problems when transporting this idea into the bitstream framework? A multiple root of f (say, of multiplicity ℓ) transforms into up to ℓ roots of f^*, real or non-real, simple or multiple. Still, all these roots are in a $9\sqrt[n]{\mu}$ neighborhood around the multiple root of f, and if the intervals are sufficiently large, all roots of f^* arising from the multiple root are counted.

Definition 2.6.7 (long interval). An interval is called *long* if its width is at least $18n\sqrt[n]{\mu}$.

Theorem 2.6.8. *For a long interval I with midpoint m_I, it holds that*

$$\#\text{ of roots of } f \text{ in } U_{\mathrm{w}(I)/2n}(I) \leq \operatorname{Var}(f^*, I^+) \leq \#\text{ of roots of } f \text{ in } U_{6n\mathrm{w}(I)}(m_I).$$

Proof. See [MS09, Theorem 14]. □

Thus, for a long interval I containing a root of f of multiplicity ℓ, $\operatorname{Var}(f^*, I^+) \geq \ell$, so we can not "lose" a multiple root of f when subdividing with respect to f^*, even if all roots become non-real. The next lemma states that simple roots of f are basically detected as in the square-free case.

Lemma 2.6.9. *Let I be a long interval with $\operatorname{Var}(f^*, I^+) = \operatorname{Var}(f^*, I) = 1$. Then I' (as defined in Theorem 2.6.3) is an isolating interval for a root of f.*

Proof. See [MS09, Lemma 15]. □

[19] We leave out the easy-to-handle special case where the multiple root is hit by some subdivision point.

These two results already recommend the bitstream version of the m-k-Descartes method. We approximate f by f^*, with respect to some approximation parameter μ, and subdivide the unit interval in the same manner as in Algorithm 2.20. An isolating interval I is identified if both I and I^+ have sign variation 1. Moreover, we consider all active intervals (i.e., those intervals I with $\mathrm{Var}(f^*, I^+) > 1$). If all these intervals show a sign variation of k or less, a failure indicator is returned (the k-case). Otherwise, let J denote the convex hull of all active intervals, that is, the smallest interval containing all active intervals. J is known to contain all multiple real roots of f, according to Theorem 2.6.8. If exactly $m-1$ simple roots with isolating intervals $I'_1, \ldots, I'_{m-1}$ of f have been detected, and J is disjoint from each I'_j, then J is known to be isolating, because the total number of roots is known to be m (the m-case). Once the subdivision produces an interval of width less than $18n\sqrt[n]{\mu}$, μ is squared, and the subdivision process restarts with an improved approximation polynomial f^*.

A special treatment is required if f^* vanishes at a subdivision point. It is not clear a priori whether such a root of f^* corresponds to a simple or multiple root of f. However, it is known to correspond to a simple root if both the left and right neighboring intervals I_ℓ and I_r satisfy $\mathrm{Var}(f^*, I_\ell^+) = 1 = \mathrm{Var}(f^*, I_r^+)$ ([MS09, Lemma 15]). Thus, the algorithm stores the vanishing subdivision points separately, and if such a subdivision point m has no adjacent active interval, $[m - 9\sqrt[n]{\mu}, m + 9\sqrt[n]{\mu}]$ is added to the list of isolating intervals for the simple roots.

The pseudo-code formulation is given in Algorithm 2.21 (see also [MS09, Alg.4] for an alternative formulation). We introduce a flag CHECK in the interval queue Q. We maintain the invariant that whenever CHECK is the first element in the queue, all further elements are intervals of same length. In this situation, we check whether the intervals have become short (then, we increase the precision and restart), or whether the m-case is satisfied (then, we return isolating intervals), or whether the k-case is satisfied (then, we return a failure indicator). Apart from that, the subdivision scheme is analogous to the square-free case.

It is immediately apparent that a failure of the algorithm implies that f has more than one multiple root. Indeed, if no active interval has a sign variation of $k+1$, then no root of multiplicity $k+1$ exists and thus, f has at least two multiple roots. Also, if intervals $I'_1, \ldots, I'_m$ are returned, each interval is isolating and disjointness is enforced by the algorithm, thus they form valid isolating intervals for the roots of f.

It is not clear yet that the algorithm always terminates. However, it appears quite intuitive: Once μ is sufficiently small, the complex roots of f will not affect the sign variation numbers, if intervals are of size $\approx 18n\sqrt[n]{\mu}$. Thus, in this situation, for each interval I, the multiplicity of the real roots contained in I^+ will be counted. Thus, either the k-case will be detected, or J will only consist of the interval containing the multiple root, plus some adjacent intervals, and will be disjoint from any other isolating interval. The next theorem quantifies this intuition.

Theorem 2.6.10. *Algorithm 2.21 terminates for*

$$\mu \leq \min\left\{\left(\frac{s(f)}{72 \cdot 25n^2}\right)^n, 2^{-7n}\right\}.$$

Proof. See [MS09, Theorem 17]. □

Algorithm 2.21. The (deterministic) m-k-bitstream-Descartes method

INPUT: $F \in \mathbb{R}[t]$, $m = \#$ of real roots of f, $k = \deg\gcd(f, f')$
OUTPUT: Isolating intervals for all real roots of F, or a failure indicator

procedure M_K_BITSTREAM_DESCARTES(F, m, k)
  $f \leftarrow F(4B(t - \frac{1}{2}))$ ▷ Scaling
  $\mu \leftarrow 2^{-7n}$
  Approximate each coefficient of f by $\frac{\mu\|p\|_1}{n+2} \rightsquigarrow f^*$
  $O^* \leftarrow \emptyset; CO^* \leftarrow \emptyset$ ▷ Isolating intervals and candidate output list
  $Q \leftarrow \{\text{CHECK}, (0,1)\}$ ▷ interval queue, CHECK is a special flag
  while true **do**
    $I \leftarrow$ first element of Q. Remove I from Q
    if $I = \text{CHECK}$ **then** ▷ all intervals in A have same length
      If intervals in Q are short, set $\mu \leftarrow \mu^2$ and **goto** 4
      for $[p,p] \in CO^*$ **do** ▷ Check singleton intervals for simpleness
        if no interval in Q has p as boundary, move $[p,p]$ from CO^* to O^*
      end for
      if $\mathrm{Var}(f^*, I^+)$ for all $I \in Q$, **return** a failure indicator ▷ the k-case
      $J \leftarrow$ the convex hull of all intervals in Q.
      if O^* contains exactly $m-1$ intervals $I_1, \dots, I_{m-1}$ and J is disjoint from all I_j **then**, **return** $4B(I_1' - \frac{1}{2}), \dots, 4B(I_{m-1}' - \frac{1}{2}), 4B(J - \frac{1}{2})$ ▷ the m-case
      end if
      Append CHECK at the end of Q
    else ▷ $I = (c,d)$ is an interval
      if $\mathrm{Var}(f^*, I^+) > 1$ **then**
        $m \leftarrow \frac{c+d}{2}$
        if $f(m) = 0$, append $[m,m]$ to CO^*
        Append (c,m) and (m,d) to Q
      else if $\mathrm{Var}(f^*, J) = 1$, append I to O^*
      end if
    end if
  end while
end procedure

This bound is the essential ingredient for the complexity analysis of Algorithm 2.21, which works analogously to the square-free case (Algorithm 2.20). The simple idea is that both methods produce exactly the same subdivision tree for any fixed approximation parameter μ. One obtains (again assuming asymptotically fast Taylor shifts):

Theorem 2.6.11. *Let $f = \sum a_i t^i \in \mathbb{R}[t]$ be a polynomial with $|\mathrm{lcf}(f)| \geq 1$, and $|a_i| < 2^{\tau-1}$ for all i. In case of both success and failure, the bit complexity of Algorithm 2.20 is*

$$\tilde{O}\left(n^3\left(\tau + \log\frac{1}{\mathrm{sep}(f)}\right)^2\right).$$

The coefficients of f need to be approximated with $O(n(\tau + \log\frac{1}{\mathrm{sep}(f)}))$ bits after the binary point.

We remark that the costs for obtaining the approximation polynomial, that is, for approximating the coefficients of the bitstream polynomial f, is not counted by Theorem 2.6.11. For a concrete application, it must be analyzed separately to find out how costly it is to get $O(n(\tau + \log \frac{1}{\mathrm{sep}(f)}))$ bits of each coefficient.

Oracle-bitstream-Descartes

So far, we have not been able to isolate the real roots of (bitstream) polynomials with more than one multiple root. The m-k-Descartes method terminates with a failure in this case. We present yet another subdivision method that works for any case, but that also requires more precomputed information about the polynomial. The requirements of the method are quite specialized, and adapted to a situation arising during the topological analysis of an algebraic curve (see Section 3.2.3).

For simplicity, we first look at an integer polynomial $f \in \mathbb{Z}[x]$ again. f might have an arbitrary number of multiple roots, and our method will compute isolating intervals for its real roots, (without making f square-free beforehand). As additional input, it requires a triple $(m, \mathfrak{o}_S, s)$ satisfying the following properties

- m is the total number of real roots (as in the m-k-Descartes method)
- Let $S \subset V(f)$ be a subset of the real roots of f. $\mathfrak{o}_S$ is a function (we call it *oracle*) that maps an interval $I \subset \mathbb{R}$ to a Boolean $b \in \{0, 1\}$ and indicates whether I contains at least one root in S.
- s is the number of elements in S
- All roots of f that are not in S have odd multiplicity

Less technically, we have s special roots of f (which form the set S), and we have an oracle $\mathfrak{o}_S$ that tells us whether an interval contains a special root. All non-special roots must be of odd multiplicity.

How does this input help to isolate the roots? Assume first that $s = m$. Then, the method is trivial. Just subdivide until m disjoint intervals are found for which the oracle $\mathfrak{o}_S$ returns 1. At the other extreme, assume that $s = 0$. Then all roots must be of odd multiplicity, and the method is also simple: Subdivide until m disjoint intervals with odd sign variation are detected. Such intervals must contain a real root according Theorem 2.4.26 (Descartes' rule of signs).

In general, the method works as follows. Start the subdivision, as usual, and during the execution check each interval using the oracle $\mathfrak{o}_S$ for the presence of a special root. Stop the subdivision if there are precisely s disjoint intervals containing special roots and further $(m - s)$ disjoint intervals that all have an odd sign variation. Returns those special and odd intervals as isolating intervals.

It should not be hard to see that this method works correctly, assuming that the input has the postulated requirements. Also, observe that the method returns as soon as the algorithm has computed a collection of isolating intervals for the real roots of f (this is not the case for the usual Descartes method, because an isolating interval might be further refined due to non-real roots close that are close to the real interval).

How to transfer this into the bitstream model? The basic structure of the bitstream version is the same as in the bitstream-Descartes, or the m-k-bitstream-Descartes case. We subdivide the unit interval with respect to a (scaled) approximation polynomial f^* with approximation parameter μ, and proceed as long as all intervals are long, that is, of a width of at least $18n \sqrt[n]{\mu}$. During the subdivision, we maintain a set of active intervals. As before, an active interval I is defined by $\mathrm{Var}(f^*, I^+) > 1$. Along the way, we also detect simple intervals, that is, intervals satisfying $\mathrm{Var}(f^*, I) = \mathrm{Var}(f^*, I^+) = 1$ which are stored, but no longer subdivided. Moreover, f^* might vanish at subdivision points. We call an interval *interesting* if it is simple, active, or adjacent to a vanishing subdivision point.

The interesting intervals decompose into connected components in the obvious way (to be precise, we have to take the closure of each interesting interval, i.e., we look at $[c, d]$ instead of (c, d)). We call these components *clusters*. Obviously, each cluster is an interval again, and no root of f^* lies at the boundary of a cluster. We assign the variation v_J for each cluster J as follows. Let $I_1, \ldots, I_d$ be the interesting intervals belonging to J, and $p_1, \ldots, p_{d-1}$ the interior boundary points. Then

$$v_J := \sum_{k=1,\ldots,d} \mathrm{Var}(f^*, I_k) + \sum_{k=1,\ldots,d-1} \mathrm{mult}(f^*, p_i).$$

We stop the subdivision if some interval is subdivided to size smaller than $18n \sqrt[n]{\mu}$ (in this case, μ is squared and the method is repeated with a better f^*), or if the following success conditions are both satisfied:

1. There are exactly s clusters for which $\mathtt{o}_S$ returns 1 (i.e., the special roots are disjoint)
2. There are exactly $(m - s)$ clusters with odd variation v_J for which $\mathtt{o}_S$ returns 0

In this case, the s special clusters and the remaining $(m - s)$ odd clusters, all extended by $9 \sqrt[n]{\mu}$ to both sides,[20] are returned as isolating intervals.

Lemma 2.6.12. *If v_J is odd for a cluster J, then J', the extension of J by $9 \sqrt[n]{\mu}$ to both sides, contains a real root of f.*

Proof. Since v_J is odd, f^* has an odd number of real roots inside J (counted with multiplicity). When transforming from f^* to f, real roots stay real or can only become complex pairwisely (and vice versa). Note that J has a distance of at least $18n \sqrt[n]{\mu} > 18 \sqrt[n]{\mu}$ from any other cluster because at least one non-interesting interval must separate them and each considered interval is long. Therefore, roots within J cannot pair with roots from different clusters when transforming f^* to f. Hence, the parity of the number of real roots remains the same. For that reason, at least one root of f^* remains real, and it can only move up to $9 \sqrt[n]{\mu}$ away from J. ☐

Lemma 2.6.13. *If the success condition from above is satisfied with $J_1, \ldots, J_s$ special clusters, and $J_{s+1}, \ldots, J_m$ odd non-special clusters, then the extended intervals $J'_1, \ldots, J'_m$ (by $9 \sqrt[n]{\mu}$ to both sides) are disjoint, isolating intervals of f.*

Proof. With the same argument as before, the extended intervals are disjoint, because different clusters have a distance of at least $18 \sqrt[n]{\mu}$ from each other. Moreover, each special cluster contains a root of f, by the properties of $\mathtt{o}_S$, and each non-special odd cluster contains a root by the previous lemma. Thus, we found m disjoint isolating intervals. ☐

Note that more clusters might exist when the success condition is met, and yet real roots of f^* might be contained therein. However, it is certain that those are caused by non-real roots of f and thus do not have to be further considered.

[20] The extension of the clusters is not always necessary. One only has to extend non-special clusters that consist only of a simple interval. Still, we extend all clusters for simplicity.

Algorithm 2.22. The (deterministic) oracle-bitstream-Descartes method

INPUT: $F \in \mathbb{R}[t]$, $m = \#$ of real roots of f, $\mathfrak{o}_S$ to identify special roots, $s = \# of special roots$

OUTPUT: Isolating intervals for all real roots of F

procedure ORACLE_BITSTREAM_DESCARTES$(F, m, \mathfrak{o}_S, s)$
    $f \leftarrow F(4B(t - \frac{1}{2}))$ ▷ Scaling
    $\mu \leftarrow 2^{-7n}$
    Approximate each coefficient of f by $\frac{\mu\|p\|_1}{n+2} \rightsquigarrow f^*$
    $O^* \leftarrow \emptyset; CO^* \leftarrow \emptyset$
    $Q \leftarrow \{\text{CHECK}, (0,1)\}$ ▷ interval queue, CHECK is a special flag
    while true **do**
        $I \leftarrow$ first element of Q. Remove I from Q
        if $I = \text{CHECK}$ **then** ▷ all intervals in A have same length
            If intervals in Q are short, set $\mu \leftarrow \mu^2$ and **goto** 4
            Form disjoint clusters $J_1, \ldots, J_k$ with respect to $Q \cup O^*$
            if $\mathfrak{o}_S(J_i) = 1$ for s clusters $J_{i_1}, \ldots, J_{i_s}$ and $\mathfrak{o}_S(J_i) = 0 \wedge v_{J_i}$ is odd for $m - s$ clusters $J_{i_{s+1}}, \ldots, J_{i_m}$ **then**
                return $4B(J'_{i_1} - \frac{1}{2}), \ldots, 4B(J'_{i_m} - \frac{1}{2})$ as isolating intervals
            end if
            Append CHECK at the end of Q
        else ▷ $I = (c, d)$ is an interval
            if $\text{Var}(f^*, I^+) > 1$ **then**
                $m \leftarrow \frac{c+d}{2}$
                if $f(m) = 0$, append $[m, m]$ to CO^*
                Append (c, m) and (m, d) to Q
            end if
            if $\text{Var}(f^*, J) = 1 = \text{Var}(f^*, I^+)$, append I to O^*
            if $\text{Var}(f^*, J) = 0$ and $\text{Var}(f^*, I^+) = 1$ and I has a boundary point in CO^*, append I to O^*
        end if
    end while
end procedure

The pseudo-code for the oracle-bitstream-Descartes method is given in Algorithm 2.22. As for the m-k-bitstream-Descartes method, a special flag CHECK is stored in the interval queue. It is ensured in that way that the termination condition is checked whenever all active intervals have the same width. The set O^* does not only store isolating intervals for simple roots, as in the m-k-case, but also non-active intervals that have a root of f^* as one of its endpoints.

We have already argued above that the returned intervals are correct if the success condition is met. We have to argue that this condition will always be met for a good enough approximation. We can use the same bound as in the m-k-Descartes case. We define the threshold μ_0 to be

$$\mu_0 := \min\left\{\left(\frac{s(f)}{72 \cdot 25n^2}\right)^n, 2^{-7n}\right\}$$

We will see that the algorithm computes isolating intervals for any $\mu \leq \mu_0$. What is the motivation for defining the threshold μ like this? We call an interval I *almost short*, if $L_0 \leq w(I) < 4L_0$ with $L_0 := 18n\sqrt[n]{\mu}$. If $\mu < \mu_0$, it follows that $4L_0 \cdot 6n \leq \frac{s(f)}{4}$, so

$$I \subseteq U_{6n\mathrm{w}(I)}(m_I) \subseteq U_{s(f)/4}(m_I)$$

with m_I the midpoint of I for any almost short interval I. In this situation, we have that

Theorem 2.6.14. *Let $\mu \leq \mu_0$, and let I be almost short. If $U_{s(f)/4}(m_I)$ contains a root of f of multiplicity k, $\mathrm{Var}(f^*, I^+) \leq k$. If it does not contain any root, $\mathrm{Var}(f^*, I^+) = 0$.*

Proof. This is a direct consequence of Theorem 2.6.8 and of the choice of μ_0. □

If follows that if $\mu \leq \mu_0$, and I is almost short with $\mathrm{Var}(f^*, I^+) \neq 0$, we can assign to I a unique real root α of f that causes the counting. We say that I is *triggered* by α.

Lemma 2.6.15. *Let I_1 and I_2 be two interesting intervals triggered by two different roots α_1 and α_2 of f. Furthermore, for $i = 1, 2$, let $L_0 \leq \mathrm{w}(I_i) < 4L_0$, if I_i is active, and let I_1 and I_2 be of same size, if both are active. In this case, I_1 and I_2 are not adjacent and have a distance of at least L_0 from each other.*

Proof. To distinguish cases, we first let both I_1 and I_2 be non-active. Since I_1 is interesting, $\mathrm{Var}(f^*, I_1^+) = 1$, and either $\mathrm{Var}(f^*, I_1) = 1$ or f^* vanishes at a boundary. Thus, f^* has a root in the closure of I_1. The same holds for I_2. W. l. o. g., let $\mathrm{w}(I_1) \geq \mathrm{w}(I_2)$. If I_1 and I_2 are adjacent, $I_2 \subset I_1^+$, thus $\mathrm{Var}(f^*, I_1^+) \geq 2$, which is a contradiction.

Let I_2 be active, thus $L_0 \leq \mathrm{w}(I_2) < 4L_0$ by assumption. If $\mathrm{w}(I_1) > \mathrm{w}(I_2)$, I_1 is not active by assumption, and is at least twice as large as I_2. If they are adjacent, it follows that $I_2^+ \subset I_1^+$ and thus $\mathrm{Var}(f^*, I_1^+) \geq \mathrm{Var}(f^*, I_2^+) \geq 2$, which contradicts the fact that I_1 is not active.

It remains the case that I_2 is active, and $\mathrm{w}(I_1) = \mathrm{w}(I_2)$. Assume for a contradiction that I_1 and I_2 are adjacent. Then, their midpoints have a distance of at most $4L_0 = 72n\sqrt[n]{\mu} \leq \frac{s(f)}{25}$. The same bound holds for the width of I_1 and I_2. By triangle inequality, $|\alpha_1 - \alpha_2 \leq \frac{3}{25}s(f), s(f)$, which is clearly a constradiction. □

This proves that at least one cluster exists for each root of f. We only have to argue that each root of odd multiplicity causes a cluster J with odd variation v_J.

Lemma 2.6.16. *Let $\mu \leq \mu_0$ as before, and let α be a root of f with odd multiplicity. Then, there exists a cluster J of almost short intervals triggered by α such that v_J is odd, and J' (the extension of J by $9\sqrt[n]{\mu}$ to both sides) contains α.*

Proof. If α is a simple root of f, it follows easily that the cluster triggered by α is unique, and consists either of one interval (if the corresponding root of f^* is in the interior of some subdivision interval), or of two intervals (if the root of f^* is hit by a subdivision point). In both cases, $v_J = 1$, and J' contains α.

Let α be a multiple root of odd multiplicity ℓ, and let I be the interesting interval that contains α. Clearly, $\mathrm{Var}(f^*, I^+) = \ell$, thus I lies within some cluster triggered by α. Let $\alpha_1^*, \ldots, \alpha_k^*$ denote the real roots of f^* that originate from α; the sum of their multiplicities must be odd as well. Those roots can only be contained in I itself, at the boundaries of I, or in adjacent intervals. If an adjacent interval contains one of the α_i^*'s, it is interesting, and if a boundary contains one of these roots, the adjacent interval is interesting as well. Thus, the cluster J around I counts all roots α_j^* in its variation v_J. Moreover, there cannot be any further roots of f^* in the cluster J, since this would imply that some interval in the cluster is triggered by another root of f which contradicts Lemma 2.6.15. Thus, v_J is odd and $J' \supset J$ contains α. □

The complexity analysis is analogous to the m-k-bitstream-Descartes case. Again, for a concrete μ, Algorithm 2.22 produces the same subdivision tree as the square-free version, and the same threshold on μ is used as in the m-k-version:

Theorem 2.6.17. *Let $f = \sum a_i t^i \in \mathbb{R}[t]$ be a polynomial with $|\mathrm{lcf}(f)| \geq 1$ and $|a_i| < 2^{\tau-1}$ for all i. The bit complexity of Algorithm 2.20 is*

$$\tilde{O}\left(n^3\left(\tau + \log\frac{1}{\mathrm{sep}(f)}\right)^2\right).$$

The coefficients of f need to be approximated with $O(n(\tau + \log\frac{1}{\mathrm{sep}(f)}))$ bits after the binary point.

Note that the cost for obtaining the coefficients is again not accounted here. Additionally, the costs to evaluate the "oracle" $\mathfrak{o}_S$ are not considered here, and have to be analyzed separately in the concrete application.

2.6.3. Root refinement

We revisit the problem of finding an isolating interval of width ε (strong root isolation, see Section 2.5.2) in the case of a bitstream polynomial f. We consider two different scenarios: first, we consider the complexity of refining all real roots of f to width ε (in analogy to Section 2.5.2). Second, we analyze the problem when refining only a subset of the roots of f.

Our methods will especially apply for all variants of the Descartes method for bitstream polynomials, that is, the usual bitstream-Descartes method for square-free polynomials (Algorithm 2.20), the m-k-bitstream-Descartes method (Algorithm 2.21), and the oracle-bitstream-Descartes method (Algorithm 2.22).

Assume that an additional ε is given and that it is required that isolating intervals are of a width of at most ε. All three versions of the bitstream-Descartes method can be easily modified for this situation. The only differences are: intervals that are known to contain a simple root (formally, where $\mathrm{Var}(f^*, I) = \mathrm{Var}(f^*, I^+) = 1$) are further refined, and, if any of the algorithms reaches a state where it would return isolating intervals, it additionally checks whether all returned intervals are of a size of at most ε, and if any of them is still too large, it keeps on subdividing

Looking at the square-free case first, it is not hard to prove a threshold on μ for which intervals of size ε can be guaranteed.

Lemma 2.6.18. *Let $\varepsilon' := \frac{\varepsilon}{4B}$, where B is the root bound used in the bitstream-Descartes method (Algorithm 2.20). If the method is modified as explained above, and*

$$\mu \leq \min\left\{2^{-7n}, \left(\frac{s(f)}{63n}\right)^n, \left(\frac{\varepsilon'}{72n}\right)^n\right\},$$

isolating intervals of size ε are produced.

Proof. It follows from Lemma 2.6.4 that isolating intervals are computed for such a choice of μ. The subdivision reaches a situation where the interval widths are between $18n\sqrt[n]{\mu}$ and $36n\sqrt[n]{\mu}$. In this situation, each isolating interval satisfies

$$\mathrm{w}(I) \leq 36n\sqrt[n]{\mu} \leq 36n\frac{\varepsilon'}{72n} \leq \frac{\varepsilon'}{2}.$$

Furthermore, $9\sqrt[n]{\mu} < \frac{\varepsilon'}{4}$, and so, when extending I by $9\sqrt[n]{\mu}$ on both sides, the resulting interval I' is of width less than ε'. This interval is now scaled by $4B$ such that its width increases by the same factor and thus, the scaled interval has size less than ε. □

For the variants dealing with multiple roots, there is one complication in the analysis. Isolating intervals for non-simple roots are formed by unions of subdivision intervals (recall the clusters in the oracle method) and thus, isolating intervals might be larger than the smallest possible long interval with respect to the current approximation value μ. However, as we show next, it cannot be *much* larger than that.

Lemma 2.6.19. *Let f^* be a μ-approximation of f with*

$$\mu \leq \mu_0 := \min\left\{\left(\frac{s(f)}{72\cdot 25n^2}\right)^n, 2^{-7n}\right\},$$

let $L_0 = 18n\sqrt[n]{\mu}$, and let all subdivision intervals be of the same size and almost short. Then, each cluster of interesting intervals is of a size of at most $32L_0$.

Proof. We are in the situation to apply Theorem 2.6.14. Each interval I with $\mathrm{Var}(f^*, I^+) \geq 1$ is triggered by a unique real root α that is in a distance of at most $\frac{s(f)}{4}$ from the interval midpoint m_I.

Assume that I is an interesting interval within the cluster for α and $|m_I - \alpha| > 14L_0$, thus, there are at least three almost short intervals between m_I and α. All roots of f^* that arise out of α are either contained in the same interval or in a neighboring interval. Thus, none of them can move into the complex circle with diameter I^+. This contradicts the assumption that α triggers the interval I. It follows that the midpoint of each interval in the cluster has a distance of at most $14L_0$ from α. Thus, the cluster can not expand more than $16L_0$ to either side, that yields $32L_0$ in total. □

The same proof shows that in the m-k-bitstream-Descartes method, the convex hull J of all active intervals is bounded by $32L_0$ for $\mu \leq \mu_0$.

Lemma 2.6.20. *Let $\varepsilon' := \frac{\varepsilon}{4B}$, where B is the root bound used in the m-k-bitstream-Descartes method (Algorithm 2.21) and in the oracle-bitstream-Descartes method (Algorithm 2.22). If the methods are modified as explained above, and*

$$\mu \leq \min\left\{2^{-7n}, \left(\frac{s(f)}{72\cdot 25n^2}\right)^n, \left(\frac{\varepsilon'}{64\cdot 36n}\right)^n\right\},$$

isolating intervals of size ε are produced.

Proof. The proof is analogous to the square-free case. It follows by Lemma 2.6.4 that isolating intervals are computed for such a choice of μ (or, for the m-k-bitstream-Descartes method, a failure has been reported). The subdivision reaches a situation where the interval widths are between $18n\sqrt[n]{\mu}$ and $36n\sqrt[n]{\mu}$. In this situation, each isolating interval satisfies

$$\mathrm{w}(J) \leq 32 \cdot 36n\sqrt[n]{\mu} \leq \frac{\varepsilon'}{2}.$$

Furthermore, $9\sqrt[n]{\mu} < \frac{\varepsilon'}{4}$, and so, when extending I by $9\sqrt[n]{\mu}$ on both sides, the resulting interval I' is of a width less than ε'. This interval is now scaled by $4B$ such that its width increases by the same factor and thus, the scaled interval has a size less than ε. □

Theorem 2.6.21. *Let $\varepsilon > 0$, let $f = \sum a_i t^i \in \mathbb{R}[t]$ be a polynomial with $|\mathrm{lcf}(f)| \geq 1$, and let $|a_i| < 2^{\tau-1}$ for all i. The bit complexity of computing isolating intervals of width at most ε using an adaption of Algorithm 2.20, 2.21, or of 2.22 is*

$$\tilde{O}\left(n^3\left(\tau + \log\frac{1}{\mathrm{sep}(f)} + \log\frac{1}{\varepsilon}\right)^2\right).$$

The coefficients of f need to be approximated with $O(n(\tau + \log\frac{1}{\mathrm{sep}(f)} + \log\frac{1}{\varepsilon}))$ bits after the binary point.

Proof. We can mainly use the same complexity analysis as in [MS09]. For a concrete μ, the size of the subdivision tree is easily seen to be $O(n\log\frac{1}{18\sqrt[n]{\mu}}) = O(\frac{1}{\mu})$, similar to [MS09, Lemma 12]. Furthermore, f^* has coefficients of a bitsize of at most $O(n\tau + \log\frac{1}{\mu})$, and in each node of the tree, the bitsize grows by n bits. This yields a maximal bitsize of $O(n\tau + \log\frac{1}{\mu})$ and with $\tilde{O}(n)$ arithmetic operations per node, one obtains

$$\tilde{O}(\log\frac{1}{\mu}n(n\tau + \log\frac{1}{\mu})) \tag{2.5}$$

as the cost for a particular μ. The iteration ends for $\log\frac{1}{\mu} = O(n(\log n + \tau + \log\frac{1}{\mathrm{sep}(f)} + \log\frac{1}{\varepsilon}))$. The result follows by substituting this equation into (2.5). □

If we do not want to refine all roots to width ε but only a selection of k roots, the complexity basically decreases from $\tilde{O}(n^3\cdots)$ to $\tilde{O}(kn^2\cdots)$. The algorithmic approach suggests itself: refine each of the k isolating intervals until they are of size smaller ε. However, there is a slight complication: it might happen that an isolating interval (formed by a cluster or a convex hull of active intervals) splits into several parts after further subdivision. This can happen if some active interval in the cluster also counts roots of f^* that arise from non-real roots of f. (our assumption that each cluster is triggered by a unique root of f is only used for the worst-case analysis – practically, the algorithm might terminate with some $\mu > \mu_0$, for which this property is not guaranteed). In this case, one has to simultaneously refine all active parts until all parts that were caused by non-real roots of f finally vanish.

Considering the complexity analysis of the isolation process, we bound its complexity by assuming that μ reaches a value where each active interval is triggered by a real root of f. Thus, if we initially assume this complexity for root isolation, we can assume that all occurrences of such split events of isolating intervals are already accounted for by this bound.

Theorem 2.6.22. *Let f be a polynomial with roots $\alpha_1, \ldots, \alpha_s$, and $\varepsilon > 0$. To compute isolating intervals of width of at most ε for a selection of k real roots of f requires*

$$\tilde{O}\left(n^3\left(\tau + \log\frac{1}{\operatorname{sep}(f)}\right)^2 + kn^2\left(\log\frac{1}{\varepsilon}\right)^2\right).$$

The coefficients of f need to be approximated with $O(n(\tau + \log\frac{1}{\operatorname{sep}(f)} + \log\frac{1}{\varepsilon}))$ bits after the binary point.

Proof. The isolation of the roots determines the cost of the first summand. The subsequent refinement of k roots causes up to $k\log\frac{1}{\varepsilon}$ additional leaves. Because the arithmetic costs per node are $\tilde{O}(n)$, and the maximal bitsize is bounded by $n(\tau + \log\frac{1}{\operatorname{sep}(f)} + \log\frac{1}{\varepsilon})$, the result follows. ☐

2.6.4. Multiplicity of an isolating interval

We have learned about isolation and refinement techniques for polynomials with multiple roots. Do these methods also provide information about the multiplicities of the roots they compute? The answer is yes, partially. We can easily assign a multiplicity to each isolating interval which forms an upper bound for the multiplicity of the unique contained root. We will also see that, for a sufficiently good approximation, this multiplicity will correspond to the exact multiplicity of the root.

The isolating intervals that are returned by any of our bitstream-Descartes variants are always extensions by $9\sqrt[n]{\mu}$ on both sides of an interval J. There are several possibilities for J:

- J is an interval for which $\operatorname{Var}(f^*, J) = 1$ and $\operatorname{Var}(f^*, J^+) = 1$, thus, it corresponds to a simple root. Consequently, we set $m_J := 1$.
- J is a cluster consisting of several subdivision intervals $I_1, \ldots, I_t$ with respect to an approximation polynomial f^*. In this case, set $m_J := \max_{j=1,\ldots,t}\operatorname{Var}(f^*, I_j^+)$.
- J is the convex hull of all active intervals $I_1, \ldots, I_k$. In this case again, set $m_J := \max_{j=1,\ldots,t}\operatorname{Var}(f^*, I_j^+)$.

Lemma 2.6.23. *Let α be the root of f represented by the isolating interval I', and let m denote its multiplicity. Then $m_{I'} \geq m$.*

Proof. The statement is trivial for simple isolating intervals, thus, we assume that I is formed by the union (or convex hull) of active intervals in the subdivision. There is some active subdivision interval I_α contained in I which contains α. From Theorem 2.6.8, $\operatorname{Var}(f^*, I_\alpha^+) \geq m$, and thus $m_{I'} \geq m$ follows. ☐

Lemma 2.6.24. *Let*

$$\mu \leq \mu_0 := \min\left\{\left(\frac{s(f)}{72\cdot 25n^2}\right)^n, 2^{-7n}\right\},$$

and let f^* approximate f accordingly. Assume that J' is an isolating interval such that J is formed by intervals of width of at most $4L_0 = 72\sqrt[n]{\mu}$. Then $m_{I'} = m$.

Proof. If μ is that small, we have seen that each active interval I_j of J is triggered by some real root α of f and thus, $\operatorname{Var}(f^*, I_j^+)$ can count at most the multiplicity of the root that triggered I_j. Since all sub-intervals of J are triggered by α, it follows that $m_{J'} \leq m$. ☐

Lemma 2.6.25. *Let f be a polynomial whose roots are isolated and refined by any of the bitstream-Descartes variants. After*

$$\tilde{O}\left(n^3\left(\tau + \log\frac{1}{\operatorname{sep}(f)}\right)^2\right)$$

bit operations, the multiplicity of each isolating interval corresponds to the multiplicity of the contained root of f.

Proof. This is clear by the bound on μ given in the previous lemma. □

Note that we do not aim for an algorithm that actually returns the multiplicity of any roots (it even appears impossible to write down such an algorithm without further knowledge about the polynomial, because a lower bound of the separation of f is required). Instead, we will design an algorithm that refines all isolating intervals in a loop whose termination condition depends on the multiplicities. It is guaranteed that the loop terminates at the latest when the correct multiplicity is counted in all intervals. Lemma 2.6.25 allows us to bound the worst-case cost of such a loop.

Summary

The bitstream-Descartes method allows us to isolate the real roots of a polynomial whose coefficients are of arbitrary nature. They only need to be approximable to arbitrary precision. This allows us, for instance, to compute the fiber of a bivariate polynomial $f(x, y)$, at a non-critical x-coordinate α. Variants of the algorithm even render possible the isolation of polynomials with multiple roots, assuming the knowledge of additional information about the polynomial. The intervals thus obtained can also be refined to arbitrarily small size by further subdivision with increased precision.

Arithmetique! Algebre! Géometrie! Trinité grandiose! Triangle lumineux!
Celui qui ne vous a pas connues est un insensé!
(Arithmetic! Algebra! Geometry! Grandiose trinity! Luminous triangle! Who-ever has not known you is without sense!)

Comte de Lautréamont

3 Arrangements of Algebraic Plane Curves

We turn to the first main result of this thesis. For a set of algebraic curves, we aim for an algorithm to compute the arrangement induced by these curves.

Definition (arrangement). Given a set of input objects $s_1, \ldots, s_n \subset \mathbb{R}^2$, define for $p \in \mathbb{R}^2$ the set $o(p) := \{s_i \mid p \in s_i\}$. The *(planar) arrangement* $\mathcal{A}(s_1, \ldots, s_n)$ is the subdivision of the plane into connected point sets with invariant $o(p)$. The components (or *cells*) of an arrangement of dimension 0, 1, and 2 are called *vertices*, *edges*, and *faces*, respectively.

If the input objects are algebraic curves, a point p belongs to a face if and only if $o(p) = \emptyset$. If the input curves are non-overlapping, that is, each pair of curves intersects in finitely many points, a point p belongs to an edge if and only if $|o(p)| = 1$.

We show how to implement the basic geometric operations on points and segments needed by the generalized version of the sweep-line method of Bentley and Ottmann [BO79] to compute the arrangement of segments of algebraic curves of arbitrary degree. Our approach produces the exact result in all cases, including all degeneracies, following the EGC paradigm, but we also aim for efficiency, using a judicious combination of symbolic and adaptive-precision numeric computations.

We realize the geometric primitives by reducing them to two types of a geometric-topological analysis: the *curve analysis* computes the fiber of a curve at critical positions and the connections between points on different fibers. Its output yields a segmentation of the curve, as defined in Definition 2.2.13. The *curve pair analysis* computes the intersection points of two curves and the vertical ordering of the fiber points at critical positions. These analyses are closely related to the cylindrical algebraic decomposition of $\mathbb{R}^2$ with respect to one curve or two curves (see Section 1.2).

A well-known problem in real algebraic geometry is the *topology computation* of an algebraic curve. The goal is to compute an embedded straight-line graph G which is isotopic (Definition 2.1.5) to the curve (equivalently, to compute an isocomplex for the curve; compare Definition 2.1.8). The curve analysis constitutes a solution to this problem, but it computes more than "just" the topology. Geometric information on the curve is also available (for instance, the position of critical points), and the curve can be approximated to any precision, which is not true for most algorithms that compute the topology of curves. The output can be considered as a stable isocomplex for the input curve, according to Definition 2.1.8.

A complete complexity analysis of both the curve analysis and curve pair analysis is given. For integer curves of degree n, whose coefficients are of a bitsize of at most 2^τ, we prove a complexity bound of

$$\tilde{O}(n^{10}(n+\tau)^2).$$

This matches the best bound on topology computation that is currently known. Therefore, we show that our algorithmic approach does not worsen the complexity. although it mainly relies on the isolation and refinement of algebraic numbers to reduces the amount of symbolic computations and although additional geometric information is computed,

We implemented our algorithm and compared it to other approaches for computing *cad*s and topology. All details concerning the implementation and the experiments are discussed in Chapter 4.

Our algorithm also accepts segments of algebraic curves as input instead of whole curves. However, in this chapter, we will concentrate on complete curves as input for simplicity. We assume that the input curves are given in implicit form, that is, by their defining bivariate polynomial.

Related work: Arrangements are ubiquitous in computational geometry, and certainly one of the best studied objects in the area. See the survey articles [AS00] [Hal97] for a comprehensive overview of theoretical results, and of various applications in robotics, molecular modeling, geometric optimization, and many more.

EGC algorithms for special cases of algebraic curve have been extensively considered. Sweep-line-based implementations exist for circular arcs [DFMT02] [WZ06], conics [WZ06] [EKP+04], and Bezier curves [HW07]. Also, other special cases such as quartic curves [CGV07] and non-singular curves of any degree [KCMK00] [Wol03] have been considered. The strategy of reducing arrangement computation to the analysis of curves and curve pairs has already been presented in [EKSW06] and analyses for the case of cubic curves are provided. This approach was implemented as part of the EXACUS library [BEH+05]. Later on, the same strategy was succesfully applied to compute arrangements of other curve types in EXACUS: conics ([BEH+02] discusses an older version) and projections of quadric intersections [BHK+05].

For the general case, Milenkovic and Sacks [MS07] compute an approximate solution and prove, under certain assumptions about the underlying numerical solver, that there exists a perturbation of the input which realizes the computed arrangement. Alberti et al. [AMW08] give a solution that can handle both implicitly represented curves and parametric curves whereas our approach is restricted to implicitly defined curves.[21] Their approach is based on subdivision and computes the correct arrangement under the assumption of a certified multivariate root solver. This root solver has to perform symbolic computations, and thus, their approach suffers from the same practical limitations as ours for high degrees. Also, no certified implementation of their algorithm is currently provided, and a complexity analysis is missing.

[21] Parametric curves can, however, be turned into implicit curves by a technique called *implicitization* [CLO97, §3].

Computing the topology of a curve is a well-studied problem. Many approaches [BPR06, §11.6] [DET09] [SW05] [GVN02] [GVEK96] [BPR06, §11.6] apply a shear transformation (compare Definition 2.3.30) to arrive at an isotopic curve in generic position , which is easier to handle. Other approaches preserve more geometric information on the curve; see [CLP+09] [Hon96], and the approaches for computing a *cad* (with adjacency information) from Section 1.2. Almost all of the mentioned algorithms work by first determining the critical positions (either of the sheared or the original curve) – often called the *projection step*, and subsequently computing the fiber at critical positions – often called the *lifting step*). The recent approach by Cheng et al. [CLP+09] is an exception. It first identifies isolating boxes for the critical points of the curve, using a symbolic root solving technique called *rational univariate representation* [Rou99], and then subdivides the plane away from these boxes. In this way, the approach avoids computing complete fibers and is less vulnerable to degenerate situations that arise from the choice of the coordinate system, such as covertical critical points. We will compare our approach with theirs in the experimental section. Another subdivision algorithm by Burr et al. [BCGY08] avoids the use of root solvers completely: They first isolate the singular points of the curve using the so-called *evaluation bound*, and then compute a mesh outside the singular regions, using a variant of the algorithm by Plantinga and Vegter [PV07]. This approach seems promising, as it is well-suited for topology computation within a certain range, and it removes the bottleneck of all prior approaches; the symbolic computation of the critical points; however, their algorithm uses a worst-case lower-bound of the evaluation bound, which can highly overestimate the required precision to isolate the singular points (similar to the constructive separation bounds discussed in Section 1.2). As long as theoretical and experimental results on the method are missing, it is difficult to judge the quality of the algorithm.

The time complexity for topology computation has also been studied for several approaches. For a curve of magnitude (n, τ), we set $N := \max\{n, \tau\}$. Arnon and McCallum [AM88] gave the first polynomial bound of $O(N^{30})$. Gonzalez-Vega and Kahoui [GVEK96] improved this to $\tilde{O}(N^{16})$ (with classical arithmetic), and Basu et al. [BPR06, §11.6] prove $\tilde{O}(N^{14})$. The best known bound has been given by Diochnos et al. [DET09], namely $\tilde{O}(N^{12})$. Their approach is purely symbolic and makes extensive use of the Sturm sequence in various substeps. We will be able to prove the same bound for our approach, although our algorithm is also focused on practical efficiency. Cheng et al. [CLP+09] state a complexity of $\tilde{O}(N^{26})$ for their method.

Outline of this chapter. The Bentley-Ottmann sweep-line algorithm and its geometric primitives are explained in Section 3.1, as well as the reduction from the primitives to curve analysis and curve pair analysis. The algorithms to provide these analyses are then described in detail in Section 3.2 (one curve) and in Section 3.3 (curve pairs), as well as their respective complexity analyses. For both methods, we heavily use the algebraic concepts and algorithms provided in Chapter 2.

3.1. Arrangements by sweeping

We first describe the data structure that stores the output of our algorithm: the *doubly-connected edge list* (Section 3.1.1). Then, we discuss the Bentley-Ottmann sweep-line algorithm for general x-monotone segments (Section 3.1.2) and introduce the set of required geometric primitives. In Section 3.1.3, we explain how points and segments on algebraic curves can be represented, and finally in Section 3.1.4, we show how the geometric primitives of the sweep reduce to the analysis of single curves and of pairs of curves.

3.1.1. The doubly-connected edge list (DCEL)

We present the doubly-connected edge list, DCEL for short, our data structure to represent, query, and manipulate planar arrangements. Since its combinatorial aspects are not the focus of this work, we only state its main characteristics for context. A discussion of the data structure at the undergraduate level can be found in [dBvKOS00, §2.2]. Full details are provided in [Ber08] and [KC08].

A DCEL stores three types of records, namely vertices, halfedges, and faces. A halfedge represents a directed edge connecting two vertices. There always exists a *twin* halfedge that points in the other direction. Thus, vertices, twin pairs of halfedges, and faces represent 0-, 1-, and 2-dimensional cells of the subdivision, respectively. The DCEL contains methods to iterate through the list of vertices, the list of halfedges, and the list of faces. Each cell has additional data that relates it topologically to the other cells of the subdivision.

Halfedge: Each halfedge stores a pointer `twin` to its twin, a pointer `source` to its source vertex, a pointer `face` to the (unique) face that is adjacent to the halfedge and a pointer `next` to the next halfedge that is adjacent to the same face.

Vertex: Each vertex v stores a pointer `halfedge` whose source vertex is v. If no such halfedge exists, v stores a pointer `face` for its surrounding face.

Face: Each face f stores a pointer `outer` to a halfedge that is part of the bounding cycle of the face.[22] Moreover, it stores two lists for holes in the face, that is, connected components in its interior. The first list stores pointers to isolated vertices. The second stores pointers to halfedges whose face pointer points to f, and such that each halfedge lies on a disjoint cycle that bounds a hole inside f.

Besides the described data, vertices, halfedges and faces can also be equipped with additional data fields. For instance, it might be useful to store the coordinates of a vertex, or sample points of (half)edges to provide more geometric information about the DCEL.

3.1.2. The Bentley-Ottmann sweep-line algorithm

We revise the well-known sweep-line algorithm to compute arrangements of linear or curved segments. It was first presented by Bentley and Ottmann [BO79]; a detailed report on its implementation in the LEDA library (for line segments) can be found in [MN00, §10.7].

The sweep-line algorithm requires its input to be decomposed into *x-monotone segments* as defined in Definition 2.2.12. Note that this definition allows vertical segments, unbounded segments, and isolated points as sweepable segments. Such special situations must be handled by a complete approach, but we assume for simplicity in our explanation that segments are bounded and non-vertical.

Conceptually, the sweep-line algorithm sweeps the plane with a vertical line ℓ from left to right and records features of the arrangement as ℓ passes over *event points*, that is,

[22] By using the `next` pointer of halfedges, the whole cycle is accessible.

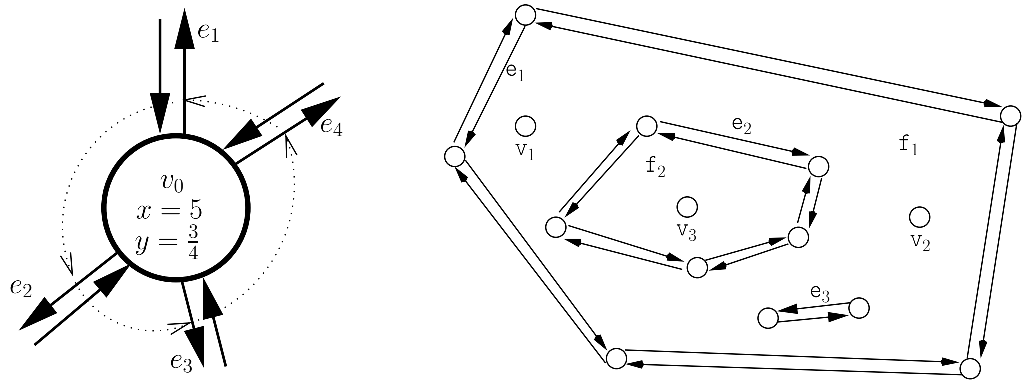


Figure 3.1. On the left: Schematic view of a DCEL vertex v_0, having an additional data field for its geometric embedding in $\mathbb{R}^2$. The halfedges with source vertex v_0 are denoted by $e_1, \ldots, e_4$. Assume that `halfedge`$(v_0) = e_1$. To iterate through the halfedges starting in v_0, one repeatedly calls `next(twin(e))`, starting with e_1. Note that this indeed yields the sequence e_1, e_2, e_3, e_4.
On the right: Schematic view of a complete DCEL structure. We have `outer`$(f_1) = e_1$, and the bounding cycle can be traversed by repeatedly calling `next`(e), starting with e_1. Moreover, f_1 has a list v_1, v_2 of isolating vertices (v_3 is not inside f_1 but appears in the isolating list of f_2) and a list of halfedges e_2, e_3 denoting the inner bounding cycles.

points at which segments start, intersect, or end. For that, the algorithm processes event points in lexicographic order and maintains the following invariant: On the left of ℓ, the arrangement has already been constructed. On ℓ, we store in a structure called the *status line*[23] a sorted sequence of intersections with segments. The half-plane right of ℓ is yet unexplored. The *event queue*[24] stores, in lexicographic order, some of the future event points; at least those at which segments start, end, or segments adjacent on the sweep line intersect. In particular, the next event point is always contained in the event queue. Processing it requires updating the status line, which means remvoing segments ending at the event, changing the order of segments intersecting in the event, and adding segments starting at the event. Moreover, the DCEL structure must be updated.

The sweep-line procedure is summarized in Algorithm 3.1. It does not directly compute the arrangement $\mathcal{A}(s_1, \ldots, s_n)$ but only lists all event points and the segments involved in each event. However, the creation of a DCEL structure becomes relatively straightforward when all event points are known (this can be done, for instance, by a second sweep through the plane, or by integrating the necessary steps directly into Algorithm 3.1). We omit further details of this step. Instead, we describe in more detail how an event point is processed during the sweep; see Algorithm 3.2. The realizations of the single steps reduces to a few geometric predicates and constructions:

Compare_xy: Given points p and q, compare them lexicographically (required, e.g., when adding points into the event queue).

Compare_y_at_x: Given a segment s and a point p within the x-range of s, deter-

[23] also called Y-structure
[24] also called X-structure

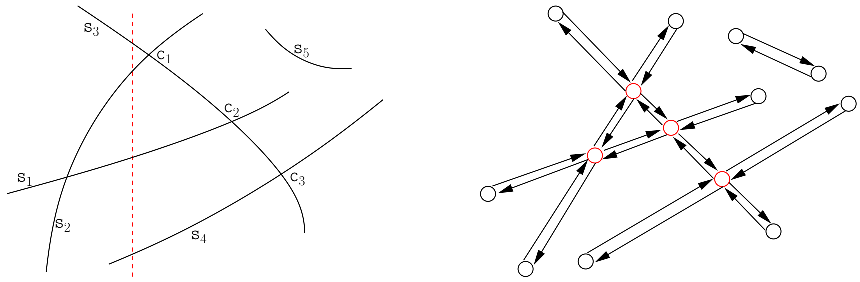


Figure 3.2. **On the left:** When the sweep-line is at the dashed position, the status line is the ordered list s_4, s_1, s_2, s_3. The event queue contains at least the endpoints of s_5, the right endpoints of $s_1, \ldots, s_4$, and the intersection points c_1 and c_2 in lexicographic order. The intersection point c_3 might or might not be contained in the event queue, since s_3 and s_4 are not adjacent in the status line.
On the right: The resulting DCEL produced by the sweep-line algorithm. Intersection vertices are drawn in red.

Algorithm 3.1. Bentley-Ottmann sweep-line algorithm

INPUT: x-monotone segments $s_1, \ldots, s_n$
OUTPUT: A sequence of elements $((p_1, S_1), \ldots, (p_k, S_k))$, where $p_1, \ldots, p_k$ are the event points with respect to $s_1, \ldots, s_n$ in lexicographic order, and S_i is the set of segments involved in the event p_i.

```
procedure SWEEP(s_1, ..., s_n)
    Initialize an empty status line, and an empty output list
    Initialize the event queue by adding all endpoints of s_1, ..., s_n in lexicographic order
    while the event queue is not empty do
        Pop the next event e from the event queue
        PROCESS_EVENT(e)    ▷ This updates status line, event queue, and output list
    end while
end procedure
```

mine whether p lies above, on, or below s (required, e.g., for determining involved segments).

Compare_to_right: Given two segments s_1 and s_2 and an intersection point p, determine the y-order of the segments immediately right of p (required, e.g., for adding segments to the status line).

Intersections: Given two (non-overlapping) segments s_1 and s_2, compute all intersection points in their interior.

Providing this small set of functionality for a certain class of segments, the sweep-line algorithm works generically for such segments, without further adaptions. If the specified class of input curves has non-x-monotone elements, an additional primitive is required (Figure 3.3).

Make_x-monotone: Given a curve, decompose it into finitely many x-monotone segments (which are disjoint in their interior)

Algorithm 3.2. Handling of event point

INPUT: Event point e

OUTPUT: None, but the method updates event queue, status line and output list

```
1: procedure PROCESS_EVENT(e)
2:     Find the segments on the status line that are involved in the event
3:     Remove ending segments from status line
4:     Reorder the remaining segments on the status line according to their vertical ordering immediately right of the event
5:     Add starting segments to the status line
6:     Add intersections of newly adjacent segments to the event queue
7:     Append e, and the set of involved segments, into the output list
8: end procedure
```

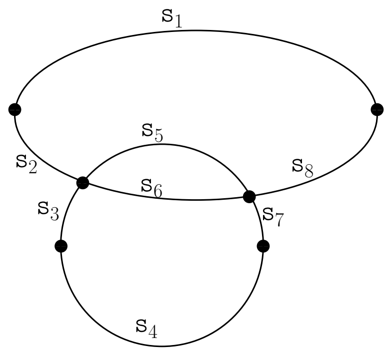


Figure 3.3. The curve, consisting of an ellipse and an intersecting circle, can be decomposed into 8 x-monotone segments $s_1, \ldots, s_8$.

Reconsider step 4 of Algorithm 3.2. Assume that k segments are intersecting in the event point p, and that p is in the interior of all segments. This step is trivial for linear segments, since the order of the segments is just reversed when passing p. For curved segments, however, it seems like $O(k \log k)$ calls of **Compare_to_right** are necessary to reorder them. But we can do better, at least for algebraic segments: Using the pairwise *intersection multiplicities* (Definition 2.3.32) of the segments, it is possible to reorder them in $O(k)$; see [BK07] for details. We therefore introduce an additional geometric primitive:

Intersection_multiplicity: Given two segments s_1 and s_2, and an intersection point p in their interior, compute the intersection multiplicity of s_1 and s_2 at p.

We have left out vertical and unbounded segments so far, but their treatment can be integrated in the above algorithm. For vertical segments, the status line can be extended such that it does not only store segments currently intersecting the sweep-line, but also segments currently overlapping it. Unbounded segments can be conceptually clipped by a *bounding box*, which can either be explicit or symbolic. Because of these special kinds of segments, the set of geometric primitives needs to be extended slightly; for instance, one has to compare unbounded segments "at infinity". The additional primitives do not pose any fundamental problem, and make the description more lengthy; therefore, we leave out the complete description here; see [BFH$^+$07] for a full list of requirements.

3.1.3. Representation of points and segments

In computational geometry, points are usually represented explicitly by their Cartesian coordinates. For exact computation with linear objects, it is mostly sufficient to represent those coordinates by a data type that models the rational numbers. We have already seen that this cannot suffice for the case of algebraic curves, because the critical points are algebraic numbers, and thus, irrational in general. We have discussed the isolating interval representation of algebraic numbers (Section 2.5) and several algorithms for computing with such numbers. We choose such isolating intervals to represent the x-coordinate of a point in $\mathbb{R}^2$. For y-coordinates of points on a curve, we could in principle use the same representation. However, due to our projection-based approach, such an representation is not directly available, and its computation involves more symbolic computations that we want to avoid. We use a more indirect representation of y-coordinates that contains enough information for our purposes.

Definition 3.1.1 (representation of points). A point $p = (\alpha, \beta)$ on a curve $V(f)$ with $f \in \mathbb{Z}[x, y]$ is represented by a triple (α, f, J), where α is represented by a pair (r, I) ($r \in \mathbb{Z}[t]$ is the defining polynomial and is the I isolating interval for α and r), and J is an isolating interval for β and $f(\alpha, y) \in \mathbb{R}[y]$ (i.e., J contains β and no other root of $f(\alpha, y)$).

An equivalent formulation is to represent p as the unique solution of the triangular system of equations

$$\begin{aligned} r(x) &= 0 \\ f(x, y) &= 0 \end{aligned}$$

inside the box $I \times J$. Note that this identifies the point uniquely, and by refining both I and J, one can approximate p to an arbitrary precision.

Once we have defined how to store points on a curve, (bounded) x-monotone segments (which are parts of an algebraic curve) become relatively straightforward. They are represented by their endpoints and by some integer i denoting that the segment is, immediately to the right-hand side of its left endpoint, the i-th segment of f from below. However, we will restrict ourselves to segments with x-ranges where the supporting curve is delineable (compare Theorem 2.2.10). That means that if the segment started as the i-th segment from below, it remains the i-th segment of f until it reaches the right endpoint. We call i the *arc number* of the segment.

Definition 3.1.2 (representation of segments). Let s be an x-monotone segment of $V(f)$ with x-range $I = [x_\ell, x_r]$, such that f is delineable over I. Then, s is be uniquely represented by the quadruple (p, q, f, i), where p and q are representations of the left and right endpoint of s, $f \in \mathbb{Z}[x, y]$ is the defining polynomial of the curve s belongs to, and the *arc number* $i \in \mathbb{N}$ denotes that s is the i-th segment of f over I.

Restriction to segments with a constant arc number as above results in cutting segments "unnecessarily" whenever the segment encounters a critical point of f in its fiber. For instance, in Figure 3.4, we count 14 segments in total, although the scenery could also be described by 8 x-monotone segments. Still, such a representation simplifies the realization of the required geometric primitives. For a slim representation of the arrangement, a post-processing step could remove unnecessary cuts afterwards.

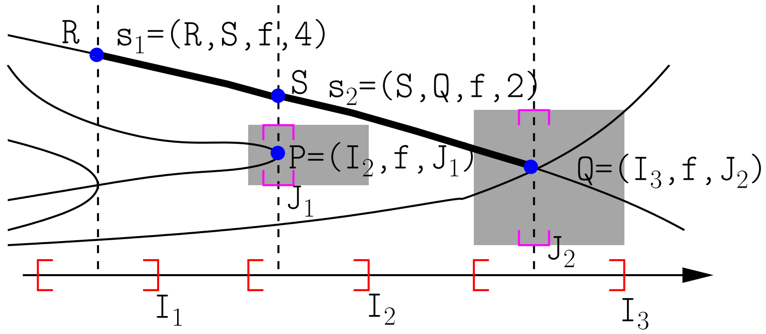


Figure 3.4. Illustration of the representation of points and segments. Note that for P and Q, the isolating boxes are also depicted in gray.

What about unbounded segments? In principle, the same representation can be chosen – we only have to introduce symbolic endpoints at infinity. We distinguish several cases. Recall from Section 2.2.3 that unbounded segments are either unbounded in x-direction, or they are vertically asymptotic segments.

- If the left end of the segment is unbounded in x-direction, its left endpoint is set to the symbolic value $-\infty$. If its right end is unbounded in x-direction, its right endpoint is set to $+\infty$.
- If a segment's end is vertically asymptotic (on either side) for $x = \alpha$, then the corresponding endpoint of the segment is set to the symbolic value $(\alpha, +\infty)$, or $(\alpha, -\infty)$, depending on whether the segment goes towards $+\infty$ or $-\infty$.

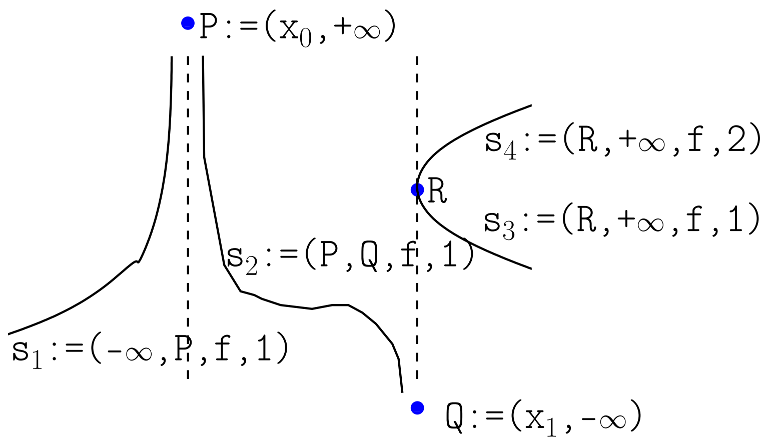


Figure 3.5. Representation of unbounded segments

Although all segments that are unbounded in x-direction formally have the same left (right) endpoint, one should not interpret this as an "intersection at infinity" in some projective or compactified sense.

3.1.4. Curve analysis and curve pair analysis

We turn to the realization of the required geometric primitives for the sweep-line algorithm, as listed in Section 3.1.2, in the case of algebraic segments represented as above. We start with the discussion of the **Make_x_monotone** primitive, that is, a curve shall be decomposed into x-monotone segments. There is no unique choice for such a segmentation; we fix the following *standard segmentation*.

Definition 3.1.3 (standard segmentation). Let $f \in \mathbb{Z}[x,y]$ be square-free and $C := \{\alpha_1, \ldots, \alpha_s\}$ be the critical x-coordinates of $V(f)$ (Definition 2.3.9). The *standard segmentation* is the segmentation (Definition 2.2.13) of $V(f)$ with respect to C.

In other words, the standard segmentation of $V(f)$ is the set of x-monotone segments, induced by the connected components of

$$V(f) \setminus \bigcup_{i=1,\ldots,s} P_i,$$

where $P_i := V(f(\alpha_i, y))$ is the fiber of $V(\mathrm{pp}(f))$ ($\mathrm{pp}(f)$ is the primitive part of f) at α_i. The x-range of each segment is either a singleton, or an open, half-open, or closed interval formed by two consecutive elements of the sequence $\{-\infty, \alpha_1, \ldots, \alpha_s, +\infty\}$. Recall from Corollary 2.2.15 that the number of segments is bounded by n^3.

Having fixed a segmentation, we can define the number of branches of a point on the curve, in a combinatorial way.

Definition 3.1.4 (branch numbers). Let $f \in \mathbb{Z}[x,y]$ be square-freeand $p \in V(f)$. The *branch numbers* $(\ell, r) \in \mathbb{N} \times \mathbb{N}$ of p in f are defined as follows. If p lies in the interior of some segment, then $(\ell, r) := (1,1)$. Otherwise, set ℓ to the number of non-vertical segments in the standard segmentation whose right endpoint is p and set r to the number of non-vertical segments whose left endpoint is p.

The branch numbers are actually independent of the chosen segmentation. They can also be defined analytically as the number of disjoint paths (either from the left- or right-hand side) converging to the point p.

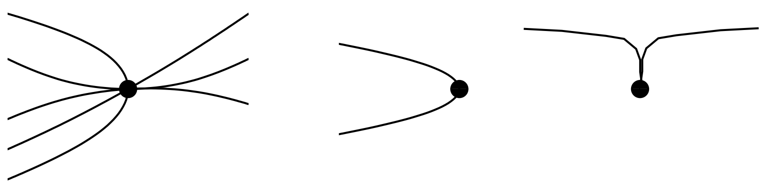


Figure 3.6. Points with branch numbers $(5,3)$, $(2,0)$, and $(1,1)$ (from left to right)

Lemma 3.1.5. *The branch numbers of a non-singular point are either* $(1,1)$ *or* $(2,0)$ *or* $(0,2)$. *The branch numbers of a non-critical point are* $(1,1)$.

Proof. Around a non-singular point, the curve can be parameterized as a function graph, in x or in y, by Theorem 2.2.8 (implicit function theorem). Thus, it has precisely two adjacent segments. If the point is non-critical, the curve is parameterizable in x, so one segment is on the left and one segment is on the right. □

The fibers P_i, as defined in Definition 3.1.3, are crucial for computing the standard segmentation. We will store these fibers together with the branch numbers of the points in the fiber and some additional information in a so-called f-stack.

Definition 3.1.6 (f-stack). Let $f \in \mathbb{Z}[x, y]$ be square-free, and let $\mathrm{pp}(f)$ denote its primitive part. For an algebraic number α, the *f-stack* at α contains the following data:

is_vertical : A flag to denote whether $V(f)$ has a vertical line at α.
fiber : A list $p_1, \ldots, p_m$ of points representing the fiber of $\mathrm{pp}(f)$ at α.
branch_numbers : A list of pairs $(\ell_1, r_1), \ldots, (\ell_m, r_m)$, where (ℓ_i, r_i) are the branch numbers for p_i.
Vertical_asymptotes : A quadruple of integers $(\ell_-, \ell_+, r_-, r_+)$, where ℓ_- is the number of segments in the standard segmentation whose right endpoint is $(\alpha, -\infty)$, in other words, the number of segments that have $x = \alpha$ as a vertical asymptote and approach it from the left and towards $-\infty$. Likewise, ℓ_+ is the number of segments with right endpoint $(\alpha, +\infty)$, r_- with left endpoint $(\alpha, -\infty)$, and r_+ with right endpoint $(\alpha, +\infty)$.

Definition 3.1.7 (curve analysis). Given $f \in \mathbb{Z}[x, y]$ is square-free, a *curve analysis* of $V(f)$ is an object that provides the following operations:

Critical_positions : Returns the list $\alpha_1, \ldots, \alpha_s$ of critical x-coordinates of $V(f)$ (in isolating interval representation).
Stack_at_x : For any algebraic number $\alpha \in \mathbb{R}$, return the f-stack at α.

At non-critical x-coordinates, the f-stack computation basically reduces to the computation of the fiber, since all other data is immediate (no vertical line, no vertical asymptotes, all branch numbers are $(1, 1)$). We use the term "computing a curve analysis" for computing the critical x-coordinates $\alpha_1, \ldots, \alpha_s$, and obtaining the f-stack for each α_i.

Having f-stacks for the critical positions, computing the standard segmentation, and consequently the **Make_x_monotone** primitive is a straightforward task, by iteration through the critical x-coordinates and constructing the x-monotone segments between consecutive critical x-coordinates. Such a construction is exemplified in Figure 3.8.

The complete algorithm also has to deal with special cases such as vertical segments and isolated points. Their treatment is straightforward: for any critical x-coordinate with a vertical component, we additionally construct vertical segments between consecutive fiber points. If a critical x-coordinate has no vertical component but a fiber point with branch numbers $(0, 0)$, it is an isolated vertex and we produce a degenerated segment for it. We skip the (quite lengthy) formal description in pseudo-code for the **Make_x_monotone** primitive.

Let us turn to the remaining primitives. We will reduce their realization to a data structure called *curve pair analysis* that we define next.

Definition 3.1.8 (critical x-coordinate for a curve pair). Let $f, g \in \mathbb{Z}[x, y]$. We call α a critical x-coordinate for the curve pair $(V(f), V(g))$ if α is critical for $V(f)$ or $V(g)$, or α is a root of $\mathrm{res}(f, g)$

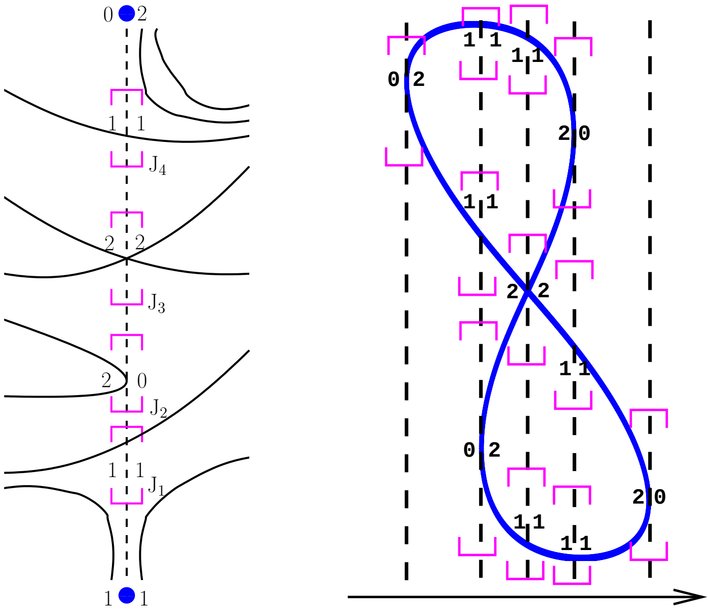


Figure 3.7. On the left: Illustration of an f-stack. The fiber is represented by the sequence $(J_1, \ldots, J_4)$ and the branch numbers by the sequence $(1,1), (2,0), (2,2), (1,1)$. The vertical asymptote numbers are $(1,0,1,2)$ in this case.
On the right: Illustration of the full curve analysis. An f-stack at each critical x-coordinate is given.

Definition 3.1.9 (fg-stack). For two curves $V(f)$ and $V(g)$, and $\alpha \in \mathbb{R}$ algebraic, the *fg-stack* at α is a string with letters "f", "g" and "i" that represents the vertical ordering of the two curves along the vertical line $x = \alpha$. More precisely, the string determines in which order the points in the fiber of $f(\alpha, y)$ and $g(\alpha, y)$ are met when the vertical line is traversed upwards. "f" stands for a point on $V(f)$, "g" for a point on $V(g)$, and "i" stands for an intersection point.

Definition 3.1.10 (curve pair analysis). Given that $f, g \in \mathbb{Z}[x, y]$ are square-free and coprime (i.e., $\deg \gcd(f, g) = 0$), a *curve pair analysis* of $(V(f), V(g))$ is an object that provides the following operations:

Critical_positions : Returns the list $\alpha_1, \ldots, \alpha_s$ of critical x-coordinates for the curve pair $(V(f), V(g))$.
Intermediate positions : Returns a list $q_0, \ldots, q_s$ of algebraic numbers such that $q_{i-1} < \alpha_i < q_i$ for $i = 1, \ldots, s$.
Stack_at_x : For any algebraic number $\alpha \in \mathbb{R}$, returns the fg-stack at α.
Intersection_multiplicity : Returns the intersection multiplicity (Definition 2.3.32). of intersection points that lie in the interior of the involved segments (compare the right of Figure 3.10).

The curve pair analysis is restricted to coprime curve pairs. If the two curves have a common component $V(h)$ (with $h = \gcd(f, g)$), one can instead consider the three curves $V(\frac{f}{h}), V(\frac{g}{h})$, and $V(h)$ that induce the same arrangement as the curve pair and that are

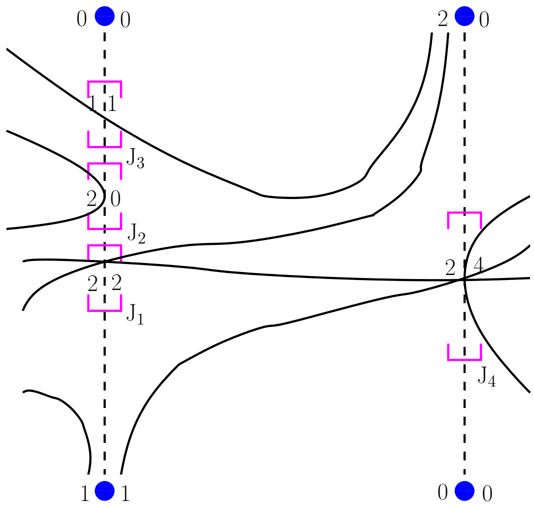


Figure 3.8. For the left-hand stack, we produce a sequence whose i-th entry contains either the isolating interval that contains the left endpoint of the i-th segment or $-\infty/+\infty$ for vertically asymptotic segments. In the example, this sequence is $(-\infty, J_1, J_1, J_3)$. The same is done for the right-hand side; one obtains $(J_4, J_4, +\infty, +\infty)$. Both sequences can be obtained combinatorially by the available information in the f-stacks. They are of the same size and directly yield the left and right endpoints for all segments between the critical x-coordinates.

pairwise coprime.

Note that the fg-stack is a purely combinatorial object (a string) and does not change in the interval between two critical x-coordinates. Therefore, computing the fg-stack for each critical x-coordinate, and for one representative point between two critical x-coordinates gives complete topological information about the curve pair. Geometric data about the curve pair can be obtained in combination with the curve analysis objects of the corresponding curves.

With a curve pair analysis at hand, all geometric primitives of the sweep-line algorithm are directly realizable:

Compare_xy : We have two points $p = (\alpha, f, I)$ and $q = (\beta, g, J)$ and want to compare them lexicographically. Let p be the i-th point in the fiber of f at α, and let q be the j-th point in the fiber of g at α. Note that i and j are easily computable using the f-stack (and the g-stack) at α, and one can even store these indices within the representation of the point when crating it.
First, we compare α and β using Algorithm 2.15 and this already yields the result except when α equals β. If $f = g$, the result is given by simply comparing i and j consider the fg-stack of $(V(f), V(g))$ that is part of their curve pair analysis and check whether the i-th occurrence of "f" in the stack comes before, after, or at the same time as the j-th occurrence of "g" (where we count "i" as an occurrence of both "f" and "g").

Compare_y_at_x : We have a point $p = (\alpha, f, I)$ and a segment $s = (q_1, q_2, g, j)$ with α in the x-range of s and want to check whether p is above, on, or below s. If α is

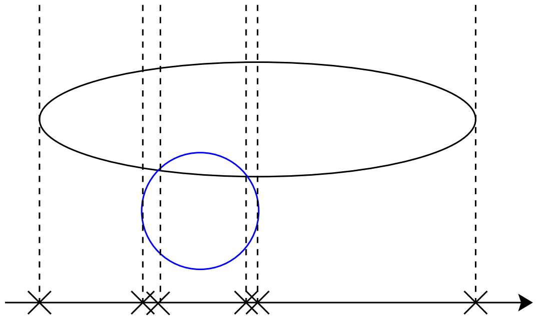


Figure 3.9. Critical x-coordinates of a curve pair.

on the boundary of the x-range of s, compare p with q_1 or q_2. If α is in the interior of the x-range, define $q := (\alpha, g, J)$, which is the point on s that is covertical to p. Call **Compare_xy** on p and q to get the result.

Compare_to_right : Given two segments $s = (p_1, q_1, f, i)$ and $t = (p_2, q_2, g, j)$ and some intersection point p with x-coordinate α, we want to determine the order of s_1 and s_2 on the right of p. Since an intersection takes places at α, it must be a critical x-coordinate for the curve pair $(V(f), V(g))$. Let β denote the intermediate position of the curve pair to the right of α. Define the points $p := (\beta, f, I)$ on s and $q := (\beta, g, J)$ on t and call **Compare_xy** for p and q.

Intersections : Given two distinct segments $s = (p_1, q_1, f, i)$ and $t = (p_2, q_2, g, j)$, we want to find all intersections in their interior. If $f = g$, there is no such intersection. Otherwise, let J denote the common x-range. If $J = \emptyset$, there is no intersection. Otherwise, consider the curve pair analysis of f and g. For each critical fg-stack within the x-range J, run through its intersection points and check for each such point p whether it lies on s and on t, using the **Compare_y_at_x** predicate. (if an intersection at a stack is found, one can proceed with the next stack).

Intersection_multiplicity : This information is directly stored in the curve pair analysis for any intersection point p in the interior of two segments.

With this approach, we reduced all geometric primitives to the curve analysis and the curve pair analysis. These objects deal with algebraic curves as a whole instead of x-monotone segments as defined in the sweep-line algorithm. This abstraction reflects the fact that algebraic computations on curves (in particular, finding their intersection points) are intrinsically global methods. This means that finding intersections of two segments in a certified way is not a significantly harder problem than finding all intersections of the two curves they belong to.

Summary

The Bentley-Ottmann sweep-line algorithm computes arrangements for arbitrary types of inputs; one “only” has to provide a rather small set of geometric primitives for the given curve type of points and segments. Using a suitable representation of such points and segments, all these primitives can be reduced to queries on two special data structures, called curve analysis and curve pair analysis.

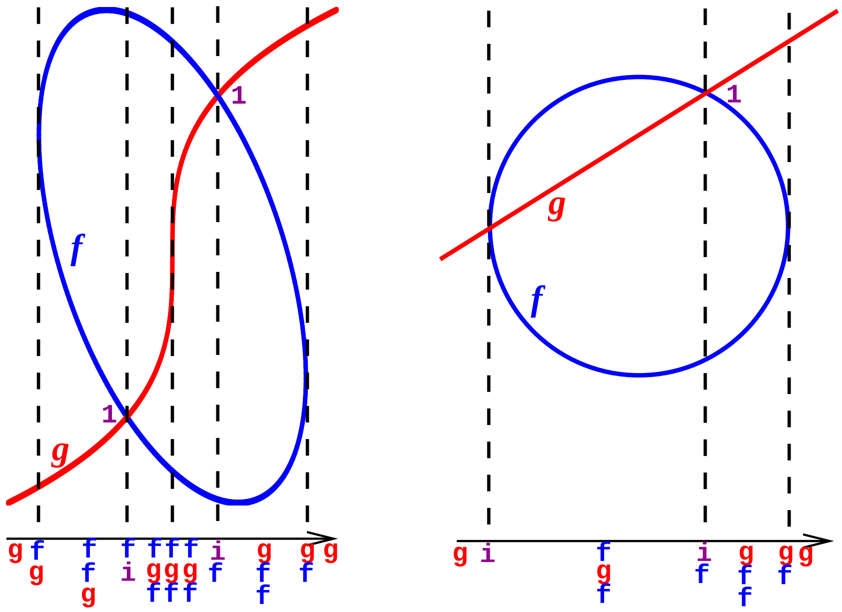


Figure 3.10. On the left: Illustration of the curve pair analysis. The numbers at the intersection points denote the intersection multiplicity.
On the right: Note that it is not required to compute the intersection multiplicity of the left intersection, since it is not in the interior of all involved segments.

3.2. Algorithm for curve analysis

We continue with the algorithm for analyzing a single curve, as defined in Definition 3.1.7. The approach (with minor modifications) was the subject of the author's Master's thesis [Ker06]; see [EKW07] for a condensed version. Different from those works, we will analyze the asymptotic bit complexity of the algorithm.

We have mentioned that many related approaches initially bring a curve into a generic position to compute its topology. Although our approach deviates from this strategy, genericity is also an important concept for our analysis. The exact definition of genericity varies slightly among the approaches, we formally define its meaning in the context of this work.

Definition 3.2.1 (generic position). A $V(f)$ with $f \in \mathbb{Z}[x,y]$ with $n = \deg_{tot}(f)$ is in *generic position*, or just *generic*, if for any $\alpha \in \mathbb{R}$, $\deg f(\alpha, y) = n$ and $f(\alpha, y)$ has at most one multiple root over $\mathbb{C}$ (which is necessarily real, since complex roots appear in pairs of complex conjugates).

Not being generic depends on the chosen coordinate system (i.e., the chosen vertical direction) rather than on the curve itself – in a different coordinate system, the curve might be in generic position. Geometrically, a generic curve has no vertical line component, no vertical asymptotic arcs, and no two critical points that are covertical. This can be used for faster lifting methods in the algorithm. In particular, for curves in generic position, the m-k-bitstream-Descartes method (Algorithm 2.6.2) can be applied to compute the fiber.

A generic position is achieved by the *shear transformation* (Definition 2.3.30) of a curve that chooses a new vertical direction

$$\mathrm{Sh}_s(f) := f(x + sy, y)$$

and the corresponding map

$$\mathrm{Sh}_s : \mathbb{R}^2 \to \mathbb{R}^2, (x, y) \mapsto (x - sy, y).$$

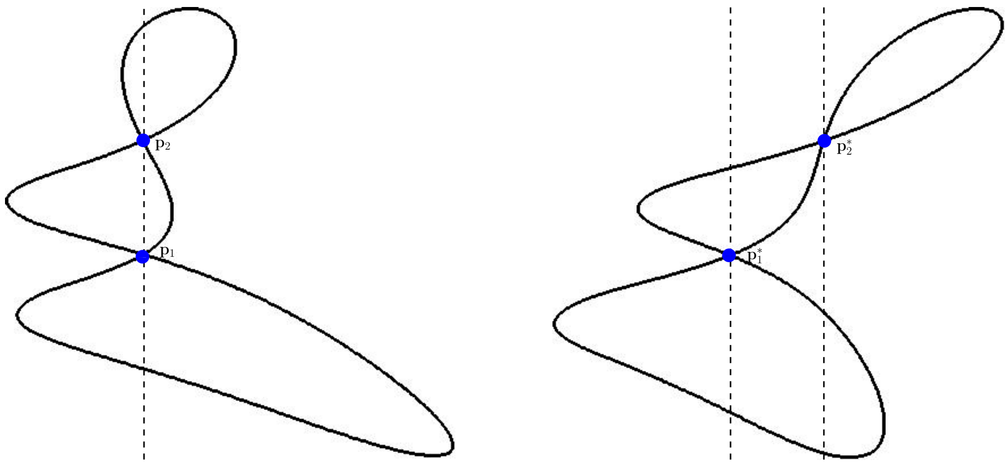


Figure 3.11. On the left: This curve is not in generic position because it has two singular points p_1 and p_2 with the same x-coordinate. **On the right**: After a shear, the curve is in generic position.

Considering a sheared curve is sufficient for topology computation , but not for our purposes – we want to represent all curves, and their analyses, in a predefined coordinate system. The reason is that we aim for an arrangement computation of curves $V(f_1), \dots, V(f_n)$; we first had to find a coordinate system that is suitable for all input curves (and all curve pairs as well), if the analyses were only performed in some sheared coordinate system.

Here is a rough overview of our algorithm for the curve analysis. It consists of two parts: The first part is an analysis in the predefined coordinate system; it always succeeds if the curve is already in generic position, but might reject a curve in non-generic position. If the direct method rejects $V(f)$, we change coordinates randomly and analyze the sheared curve until it succeeds. Afterwards, the result of the analysis in the changed coordinates needs to be transformed back into the original coordinate system. This is achieved by the second part of our algorithm; a method of analysis in the original system that always succeeds, but depends on information from a successful direct analysis in a different coordinate system.

What are the advantages of this approach? First of all, by passing to a sheared curve, we can make use of the m-k-bitstream-Descartes method for any instance, thus, a fallback to a purely symbolic method, as in many other approaches in *cad* computation, is never necessary. With the second part of our algorithm, we are still able to deduce the geometric information of the curve analysis; this is the first approach with such a back-transformation step. Also, we exploit the fact that the m-k-bitstream-Descartes detects unfortunate situations (e.g., covertical critical points) by itself; a certified precomputation of a generic direction, which would cause more symbolic computation, is prevented by a randomized choice of the shear factors.

We further proceed in this section as follows. We first investigate the direct approach in Section 3.2.1. Then, we describe how f-stacks at intermediate positions between critical x-coordinates can be computed additionally, without spoiling the complexity of the overall algorithm in Section 3.2.2. This substep is needed particularly in the indirect approach, which is the subject of Section 3.2.3. Finally, we discuss how to compute additional information not covered by the curve analysis, in Section 3.2.4.

3.2.1. The direct approach

For simplicity, we assume that the input curve f is primitive, that is, without vertical line components. Non-primitive curves are simple to handle, as we discuss subsequently in Section 3.2.4. We first try to analyze the curve directly in the predefined coordinate system, hence the name "direct method". We identify four phases of the direct method:

1. **Projection:** Computes the critical x-coordinates $\alpha_1, \ldots, \alpha_s$ by isolating the real roots of the resultant $\mathrm{res}(f, \frac{\partial f}{\partial y})$.
2. **Symbolic precomputation:** Computes some partial information about the polynomials $f(\alpha_i, y)$ for $i = 1, \ldots, s$ by exploiting the Sturm-Habicht coefficients of f.
3. **Fiber construction:** Computes the fiber of f at each α_i by using the m-k-bitstream-Descartes method.
4. **Branch numbers:** Computes the branch numbers of each fiber point over any α_i.

The second and the third steps might fail during their execution for non-generic curves, in which case the algorithm rejects the curve. Still, the approach might also get along with non-generic curves. We now describe each step of the algorithm; owing to our detailed description of algebraic tools in Chapter 2, this mainly reduces to assembling several subroutines.

Projection: We compute the sequence of principal Sturm-Habicht coefficients (Defintion 2.3.22) $\mathrm{stha}_n(f), \ldots, \mathrm{stha}_0(f)$ of our input polynomial f. Recall that those are, up to sign, equal to the principal subresultant coefficients of f and its derivative $\frac{\partial f}{\partial y}$. Thus, we compute them by Algorithm 2.5 (subresultants by pseudo-remainder). Also, recall that, by definition, the polynomial $\mathrm{stha}_0(f)$ equals, up to sign, the resultant $\mathrm{res}(f, \frac{\partial f}{\partial y})$. We can thus isolate its real roots $\alpha_1, \ldots, \alpha_s$ using Algorithm 2.10 (real root isolation).

Symbolic precomputation: From the previous step, the critical x-coordinates as well as the principal Sturm-Habicht coefficients are known. Proceed as follows for $\alpha_1, \ldots, \alpha_s$: Compute the sign of $\mathrm{lcf}_y(\alpha_i)$ (Algorithm 2.18); if the sign is zero, reject the curve. Otherwise, it is ensured that $\deg_y f = \deg f(\alpha_i, y)$; this means, in particular, that the curve has no vertically asymptotic arcs; we fill the corresponding entries in the f-stack by zeroes. Moreover, the specialization property (Theorem 2.3.17) is applicable. This means that $\mathrm{stha}_n(f)(\alpha_i), \ldots, \mathrm{stha}_0(f)(\alpha_i)$ are the principal Sturm-Habicht coefficients of $f(\alpha_i, y)$. We compute the sign of each principal Sturm-Habicht coefficient $\mathrm{stha}_j(f)(\alpha_i)$ using Algorithm 2.18. Their sign sequence reveals the numbers $m_i := \#\{\beta \in \mathbb{R} \mid f(\alpha_i, \beta) = 0\}$

(Theorem 2.3.29) and $k_i := \deg\gcd(f(\alpha_i, y), f'(\alpha_i, y))$ (Lemma 2.3.14) that will be used in the subsequent phase.

Fiber construction: We apply the m-k-bitstream-Descartes method (Algorithm 2.21) on each polynomial $f(\alpha_i, y)$, with the additional input m_i and k_i . If it fails for any instance, the curve is rejected. In the successful case, isolating intervals $\beta_{i,1}, \ldots, \beta_{i,m_i}$ are returned and one among them, say β_{i,c_i}, is distinguished from the others, since it is the only interval that might contain a multiple root. The fiber of f at α_i is stored in the f-stack by constructing points $(\alpha_i, f, \beta_{i,j})$ for $j = 1, \ldots, m_i$.

Branch numbers: We compute rational intermediate values $q_0, \ldots, q_s$ with $q_{i-1} < \alpha_i < q_i$, using Algorithm 2.14. Again, we compute for each q_i the signs of the principal Sturm-Habicht coefficients (using Algorithm 2.4) and use these signs to compute t_i, the number of points in the fiber of $V(f)$ at q_i. For a fixed α_i, let $\beta_{i,1}, \ldots, \beta_{i,m_i}$ be its fiber. For all fiber points except the distinguished one (see the previous step), we set the branch numbers to $(1, 1)$. The branch numbers of the distinguished one are set to $(t_{i-1} - m_i + 1, t_i - m_i + 1)$.

We have to argue why this branch number assignment is doing the correct thing. We have the following geometric situation. We have m_i fiber points, and there are in total t_{i-1} segments approaching from the left-hand side, and t_i segments approaching form the right-hand side. Note that none of these segments is a vertically asymptotic arc (this has been excluded in the second step of the method), thus each segment has to end in one of the fiber points. On the other hand, Lemma 3.1.5 shows that each fiber point, except the distinguished one β_{i,c_i}, consumes exactly one segment from each of the left and right. Hence, the distinguished fiber point has to absorb all remaining segments, and there are precisely $t_{i-1} - m_i + 1$ segments on the left-hand side and $t_i - m_i + 1$ on the right-hand side. Note that this simple argument relies heavily on the absence of vertically asymptotic arcs as well as on the property that there is at most one critical point of f in the fiber. The complete method is summarized in Algorithm 3.3.

Theorem 3.2.2. *If $V(f)$ is in generic position, Algorithm 3.3 computes the curve analysis of $V(f)$.*

Proof. The correctness of the algorithm follows directly by the correctness of its subroutines. We only have to argue why a curve in generic position is never rejected. Note that a curve can be rejected in only two substeps of the algorithm. The first possibility is that $\mathrm{lcf}_y(f)(\alpha_i) = 0$, which implies that $\deg f(\alpha_i, y) < \deg_y f(x, y) \le \deg_{tot} f(x, y)$, which is only possible for non-generic curves. The second possibility is that the m-k-bitstream-Descartes method fails, which is only possible if $f(\alpha_i, y)$ has at least two complex multiple roots (Section 2.6.2). Again, this is impossible for generic curves. □

Our next goal is the complexity analysis of Algorithm 3.3. We will prove the following result.

Theorem 3.2.3. *For a square-free bivariate polynomial f of magnitude (n, τ), Algorithm 3.3 requires*

$$\tilde{O}(n^{10}(n + \tau)^2)$$

bit operations (both in case of success and failure).

Algorithm 3.3. Curve analysis, direct method

INPUT: $f \in \mathbb{Z}[x,y]$ square-free, without vertical line component

OUTPUT: The critical x-coordinates of f, and an f-stack for each critical x-coordinate, or a flag `REJECT` that denotes that the curve has been rejected

procedure CA_DIRECT(f)
  Compute the principal Sturm-Habicht coefficients of f
  $\alpha_1, \ldots, \alpha_s \leftarrow$SOLVE($\mathrm{stha}_0(f)$)
  for $i = 1, \ldots, s$ **do**
    if $\mathrm{lcf}_y(f)(\alpha_i) = 0$ then **return** `REJECT`
    For each $j = 0, \ldots, n-1$, compute $\mathrm{sign}(\mathrm{stha}_j(f)(\alpha_i))$
    Compute $m_i := \#\{\beta \in \mathbb{R} \mid f(\alpha_i, \beta) = 0\}$ and $k_i := \deg\gcd(f(\alpha_i, y), f'(\alpha_i, y))$
    $\beta_{i,1}, \ldots, \beta_{i,m_i} \leftarrow$ M_K_BITSTREAM_DESCARTES($f(\alpha_i), m_i, k_i$). Let β_{i,c_i} be the distinguished output interval. If a failure occurs, **return** `REJECT`
  end for
  $q_0, \ldots, q_s \leftarrow$INTERMEDIATE($\mathrm{stha}_0(f)$)
  For each $i = 0, \ldots n-1$ and each $j = 0, \ldots s$, compute $\mathrm{sign}(\mathrm{stha}_i(f)(q_j))$
  Compute $t_i := \#\{\beta \mid f(q_i, \beta) = 0\}$
  For each $i \in \{1, \ldots, s\}$ and each $j \in \{1, \ldots, m_i\} \setminus \{c_i\}$, set the branch numbers for $(\alpha_i, f, \beta_{i,j})$ to $(1,1)$.
  For each $i \in \{1, \ldots, s\}$, set the branch numbers of $(\alpha_i, f, \beta_{i,c_i})$ to $(t_{i-1} - m_i + 1, t_i - m_i + 1)$.
end procedure

To prove this result, we consider the four phases of the algorithm separately.

Projection: Computing the principal Sturm-Habicht coefficients requires $\tilde{O}(n^7\tau)$ bit operations (Theorem 2.4.17). Note that each principal Sturm-Habicht coefficient is of magnitude $(n^2, n(\tau + \log n))$. In particular, the resultant of f and $\frac{\partial f}{\partial y}$ is of this magnitude. For isolating its real roots, its square-free part $r \in \mathbb{Z}[x]$ must be computed (as a substep within Algorithm 2.10). From Theorem 2.4.21, r has degree $p \in O(n^2)$ and a maximal bitsize of $\sigma \in O(n(n+\tau))$. Isolating its real roots requires $\tilde{O}(p^4\sigma^2) = \tilde{O}(n^{10}(n+\tau)^2)$ bit operations, according to Theorem 2.4.37.

Symbolic precomputation: Computing $\mathrm{sign}(\mathrm{lcf}_y(f)(\alpha_i))$ for each critical x-coordinate α requires $\tilde{O}(p^4(p+\sigma)^2) = \tilde{O}(n^{10}(n+\tau)^2)$ bit operations, according to Theorem 2.5.23 (of course, a better bound could be achieved here, since $\mathrm{lcf}_y(f)$ is only of magnitude (n, τ)). Secondly, we have to evaluate all the signs of $\mathrm{stha}_j(f)(\alpha_i)$. Since all $\mathrm{stha}_j(\alpha)$ are of magnitude (p, σ), we can apply Theorem 2.5.24, and achieve a complexity of

$$\tilde{O}(np^4(p+\sigma) + p^4(p+\sigma)^2) = \tilde{O}(n^{10}(n+\tau)^2)$$

also for this step.

Fiber construction: Let f_i denote the polynomial $f(\alpha_i, y)$, and let 2^{τ_i} be an upper bound for the absolute value of any coefficient of f_i. According to Theorem 2.6.11, the bit complexity of the m-k-bitstream-Descartes method is

$$O\left(n^3\left(\log n+\tau_i+\log\frac{1}{\mathrm{sep}(f_i)}\right)^2\right),$$

and $O(n(\tau_i+\log\frac{1}{\mathrm{sep}(f_i)}))$ bits of each coefficient are required. Thus, we have to bound the values τ_i and $\log\frac{1}{\mathrm{sep}(f_i)}$ in terms of n and τ. As usual, we bound their sum when adding up over all roots of r. The first one is relatively simple.

Lemma 3.2.4.

$$\sum_{i=1}^{s}\tau_i=O(n^2(\tau+n))$$

Proof. For a univariate polynomial g of magnitude (n,τ), we have for any $\alpha\in\mathbb{R}$ $|g(\alpha)|\le(n+1)2^\tau\max\{1,|\alpha|\}^n$. Since each coefficient of $f\in\mathbb{Z}[x][y]$ is of that magnitude, it follows that τ_i is bounded by

$$\tau_i\le\log(n+1)+\tau+n\log\max\{1,|\alpha_i|\}.$$

Summing over all τ_i yields

$$\sum_{i=1}^{s}\tau_i\le n^2\log(n+1)+n^2\tau+n\log\prod_{i=1}^{s}\max\{1,|\alpha_i|\}.$$

Since $\alpha_1,\dots,\alpha_s$ are roots of r (the square-free part of $\mathrm{stha}_0(f)$), it holds that

$$\prod_{i=1}^{s}\max\{1,|\alpha_i|\}\le\frac{1}{\mathrm{lcf}(r)}\mathrm{Mea}(r)\le\mathrm{Mea}(r)$$

and since r is of magnitude $(n^2,n(n+\tau))$, $\mathrm{Mea}(r)=O(n(n+\tau))$. □

For the bound on the sum of the separations, we can resort to a result by Eigenwillig [Eig08, Prop. 3.73].

Lemma 3.2.5.

$$\sum_{i=0}^{s}\log\frac{1}{\mathrm{sep}(f_i)}=O(n^3(n+\tau))$$

The proof of Lemma 3.2.5 exploits the generalized form of the Davenport-Mahler bound as given in Theorem 2.4.33. It is mostly analogous to the proof of Lemma 3.2.7 that is given later.

With these two estimates, we can bound the bit complexity of the m-k-bitstream-Descartes instances by

$$\begin{aligned}\sum_{i=0}^{s}O(n^3(\log n+\tau_i+\log\frac{1}{\mathrm{sep}(f_i)})^2) &= \tilde{O}\left(n^3(\sum_{i=0}^{s}\tau_i)^2+n^3(\sum_{i=0}^{s}\log\frac{1}{\mathrm{sep}(f_i)})^2\right)\\ &= \tilde{O}(n^3(n^2(n+\tau))^2+n^3(n^3(n+\tau))^2)\\ &= \tilde{O}(n^9(n+\tau)^2).\end{aligned}$$

Furthermore, we have to estimate the costs for obtaining $O(n(\tau_i + \log \frac{1}{\mathrm{sep}(f_i)}))$ bits of each coefficient. Let $a_0, \ldots, a_n \in \mathbb{Z}[x]$ denote the coefficients of $f \in \mathbb{Z}[x][y]$. For each $\alpha_1, \ldots, \alpha_s$, we need up to $O(n(\tau_i + \log \frac{1}{\mathrm{sep}(f_i)})) = (n^4(n+\tau))$ bits for each $a_j(\alpha_i)$, $j = 0, \ldots, n$ (actually, we only need that many bits in total for all α_i, but we do not use this). This is exactly the situation where Theorem 2.5.22 is applicable; with the notation of that theorem, $n' \leftarrow n^2$, $\tau' \leftarrow n(n+\tau)$, $k \leftarrow n$, and $L_\delta \leftarrow n^4(n+\tau)$. Thus, we obtain an upper bound of

$$\tilde{O}(n^8 + n^2(n+\tau)^2 + n^6(n^4(n+\tau) + n^2(n+\tau))) = \tilde{O}(n^{10}(n+\tau)).$$

Branch numbers: Theorem 2.5.17 proves a bit complexity of $\tilde{O}(p^4\sigma^2) = \tilde{O}(n^{10}(n+\tau)^2)$ to find the intermediate values $q_0, \ldots, q_n$. Each q_i has a bitsize of $O(p(\sigma + \log p)) = O(n^3(n+\tau))$. We have to evaluate each $\mathrm{stha}_0(f), \ldots, \mathrm{stha}_n(f)$ at each q_i, which makes up to $O(np) = O(n^3)$ evaluations, and each of them has a bit complexity of $O(pM(pp\tau)) = \tilde{O}(n^7(n+\tau))$. This makes $\tilde{O}(n^{10}(n+\tau))$ bit operations in total for this step. The assignment of the branch numbers itself has negligible complexity.

We have shown that bit complexity of the direct curve analysis is bounded by $\tilde{O}(n^{10}(n+\tau))$. If the returned value is a failure indicator, the curve is not in generic position, and we have to switch to the method for non-generic curves.

3.2.2. Intermediate stacks

Before we discuss the general approach, we need to establish an important sub-operation. Assume that the f-stacks for the critical x-coordinates are known already. In numerous situations, we need the knowledge of *intermediate stacks*, that is, some f-stack between two consecutive f-stacks at critical x-coordinates, and some f-stack at the left or right of any critical x-coordinate.

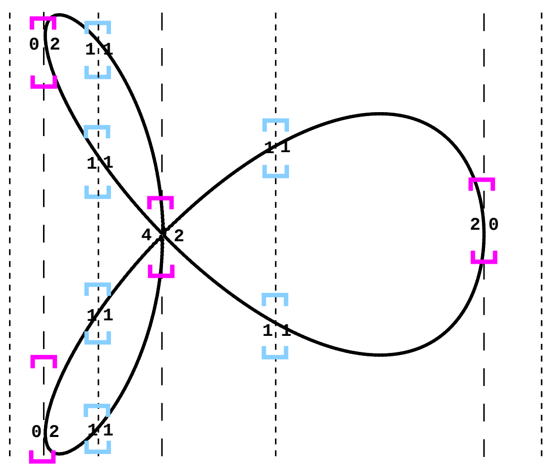


Figure 3.12. The critical f-stacks of a curve (dashed lines), and a collection of intermediate stacks (dotted).

The straightforward approach is to choose rational intermediate values $q_0, \ldots, q_n$ using Algorithm 2.14 and apply the Descartes method (Algorithm 2.9) on each $f(q_i, y)$ (note that this polynomial is square-free, since q_i is not critical). However, we fail to show a satisfying bit complexity for this approach.

Lemma 3.2.6. *Computing intermediate stacks using the intermediate values from Algorithm 2.14 has a bit complexity of*

$$\tilde{O}(n^{12}(n+\tau)^2).$$

Proof. Each intermediate value q_i has a bitsize of $O(n^3(n+\tau))$, and one can see that the sum of their bitsizes is bounded by $O(n^3(n+\tau))$ as well: The bitsize of each intermediate value can be bounded by $O(\log \frac{1}{|\alpha_i - \alpha_{i+1}|})$, where α_i and α_{i+1} are the two neighboring roots. The bound on the sum follows directly from Theorem 2.4.33 (generalized Davenport-Mahler bound). Let b_i denote the bitsize of q_i, then the polynomial $f(q_i, y)$ is of magnitude $(n, \tau + nb_i))$ and the bit complexity of all Descartes instances is bounded by

$$\tilde{O}(\sum n^4(\tau + nb_i)^2) = \tilde{O}(n^4 \sum(\tau + nb_i)^2) = \tilde{O}(n^4(n^2\tau + n\sum b_i)^2) = \tilde{O}(n^{12}(n+\tau)^2).$$

□

There is a gap of $\tilde{O}(n^2)$ compared to the complexity of the curve analysis. Although this theoretic result is by no means observable in the practical performance of the algorithm (see Chapter 4), we have to look for an alternative solution that is asymptotically no worse than the curve analysis itself. The crucial weakness in the algorithm above is that there is no algebraic relation between the intermediate values. Rolle's theorem states that between two critical x-coordinates, there is always a root of the derivative of $\mathrm{stha}_0(f)$. We will use selected roots of this derivative as intermediate values instead, because we can apply the following lemma in this situation.

Lemma 3.2.7. *Let f be of magnitude (n, τ). Let $\beta_1, \dots, \beta_s$ be roots of a polynomial $r^* \in \mathbb{Z}[x]$ of magnitude (n', τ') with $n' = O(n^2)$ and $\tau' = O(n(n+\tau))$ such that $V(f)$ is non-critical for any β_i. Set $f_{\beta_i} := f(\beta_i, y)$, and let s_i be the root separation of f_{β_i}. Then it holds that*

$$\sum_{i=0}^{s} \log \frac{1}{s_i} = O(n^3(\tau + n)).$$

Proof. Set $R := \mathrm{res}_y(f, \frac{\partial f}{\partial y})$. First of all, we can assume w. l. o. g. that r^* is coprime to R. Otherwise, consider the polynomial $r^{**} := \frac{r^*}{\gcd(R, r^*)}$. Each β_i is also a root of r^{**}, and r^{**} is still of the same magnitude.

From now, the proof is almost completely analogous to [Eig08, Prop. 3.73], under slightly simplified assumptions. Using the Davenport-Mahler bound (Theorem 2.4.33), one can see that

$$\log \frac{1}{s_i} \leq -\log \frac{\sqrt{|\mathrm{res}(f_{\beta_i}, f'_{\beta_i})|}}{\sqrt{\mathrm{lcf}(f_{\beta_i})}\mathrm{Mea}(f_{\beta_i})^{n-1}} + O(n \log n).$$

For all complex roots of r^* that are not among the $\beta_1, \dots, \beta_s$, we apply the Davenport-Mahler bound on the empty set to obtain

$$0 \leq -\log \frac{\sqrt{|\mathrm{res}(f_{\beta_i}, f'_{\beta_i})|}}{\sqrt{\mathrm{lcf}(|f_{\beta_i}|)}\mathrm{Mea}(f_{\beta_i})^{n-1}} + O(n \log n).$$

Let $\beta_1, \ldots, \beta_q$ denote all (complex) roots of r^*. We can write

$$\sum_{i=0}^{s} \log \frac{1}{s_i} = \sum_{i=0}^{q} - \log \frac{\sqrt{|\mathrm{res}(f_{\beta_i}, f'_{\beta_i})|}}{\sqrt{\mathrm{lcf}(|f_{\beta_i}|)}\mathrm{Mea}(f_{\beta_i})^{n-1}} + O(n^3 \log n)$$
$$= O(n^3 \log n) - \frac{1}{2} \log |\prod_{i=1}^{q} \mathrm{res}(f_{\beta_i}, f'_{\beta_i})| + (n-1) \log \prod_{i=1}^{q} \mathrm{Mea}(f_{\beta_i}) + \frac{1}{2}(n-1) \log \prod_{i=1}^{q} |\mathrm{lcf}(f_{\beta_i})|.$$

We bound each product separately by $O(n^3(n+\tau))$. For the first, note that $\mathrm{res}(f_{\beta_i}, f'_{\beta_i}) = R(\beta_i)$ by Theorem 2.3.5 (specialization property) and since R and r^* are coprime, none of the $\mathrm{res}(f_{\beta_i}, f'_{\beta_i})$ are zero. Consequently, using Theorem 2.3.6, we get

$$|\prod_{i=1}^{q} R(\beta_i)| \quad = \quad |\frac{\mathrm{res}(R, r^*)}{\mathrm{lcf}(r^*)^{n(n-1)}}|$$

and since $\mathrm{res}(R, r^*) \in \mathbb{Z}$, it follows that

$$-\frac{1}{2} \log |\prod_{i=1}^{q} \mathrm{res}(f_{\beta_i}, f'_{\beta_i})| \le \log |\frac{1}{\prod_{i=1}^{q} \mathrm{res}(f_{\beta_i}, f'_{\beta_i})}| \le \log |\frac{\mathrm{lcf}(r^*)^{n(n-1)}}{\mathrm{res}(R, r^*)}| \le n^2 \log |\mathrm{lcf}(r^*)|$$

and the bound of $O(n^3(n+\tau))$ is satisfied, since r^* has coefficients of bitsize $O(n(n+\tau))$ by assumption.

Likewise, since the Mahler measure is multiplicative, we can write

$$\prod_{i=1}^{q} \mathrm{Mea}(f_{\beta_i}) = \mathrm{Mea}(\prod_{i=1}^{q} f_{\beta_i}) = \frac{\mathrm{Mea}(\mathrm{res}_x(f, r^*))}{|\mathrm{lcf}(r^*)^n|} \le \mathrm{Mea}(\mathrm{res}_x(f, r^*)) \le \sqrt{n^2+1} \|\mathrm{res}_x(f, r^*)\|_\infty.$$

Thus, we have to analyze the coefficient size of $\mathrm{res}_x(f, r^*)$. The Sylvester matrix of f and r^* has up to n^2 rows with coefficients of f and n rows with coefficient of r^*. Therefore, the coefficients of $\mathrm{res}_x(f, r^*)$ cannot be larger than

$$n^2\tau + n \cdot n(n+\tau) + \log(n^2+n) = O(n^2(n+\tau)).$$

Consequently, $(n-1) \log \mathrm{Mea}(\mathrm{res}_x(f, r^*)) = O(n^3(n+\tau)$.

For the third product, we proceed similarly. Since $\mathrm{lcf}(f_{\beta_i}) = \mathrm{lcf}_y(f)(\beta_i)$, it follows that

$$\frac{1}{2}(n-1) \log \prod_{i=1}^{q} |\mathrm{lcf}(f_{\beta_i})| \le n \log |\frac{\mathrm{res}(\mathrm{lcf}(f), r^*)}{\mathrm{lcf}(r^*)^n}| \le n \log |\mathrm{res}(\mathrm{lcf}(f), r^*)|$$

and by the same argument as above, $\log |\mathrm{res}(\mathrm{lcf}(f), r^*)| = O(n^2(n+\tau))$, which proves the theorem. □

To compute the intermediate stacks, we proceed by computing the real roots of the derivative R' and selecting one such root for an interval between two consecutive critical x-coordinates (this requires the comparison of algebraic numbers; Algorithm 2.16 applied on R and R' produces an ordered list of the roots of f and f' in such a way that this selection can be made with one iteration through the list). For the leftmost and rightmost intermediate stacks, we pick $\beta_0 := -2^{\tau+1}$ and $\beta_n := 2^{\tau+1}$, respectively (compare Algorithm 2.14).

Algorithm 3.4. Constructing intermediate stacks

INPUT: $f \in \mathbb{Z}[x, y]$ square-free, f-stacks for each critical x-coordinate
OUTPUT: f-stacks at intermediate values $\beta_0, \ldots, \beta_s$. that the curve has been rejected

```
1: procedure INTERMEDIATE_STACKS(f)
2:     R ←square-free part of res_y(f, ∂f/∂y)
3:     Call MERGE_ROOTS(R, R'). Pick β_i for i = 1, ..., s − 1 such that α_i < β_i < α_{i+1}.
4:     For each i = 0, ..., s, call BITSTREAM_DESCARTES(f_{β_i})
5:     σ ← log ||R||_∞
6:     β_0 ← −2^{σ+1}, β_n ← 2^{σ+1}
7:     Call DESCARTES(f_{β_i}) for i ∈ {0, s}
8:     Construct an f-stack at β_i (for i = 0, ..., s) based on the fiber computed by the
       (bitstream-)Descartes method.
9: end procedure
```

Once the intermediate values $\beta_0, \ldots, \beta_n$ are computed, we apply the bitstream-Descartes method (Algorithm 2.20) on each $f(\beta_i, y)$ to construct the fiber. All other data in the f-stack is immediately clear (no asymptotic segments, no vertical component, and all branch numbers are $(1, 1)$).

Theorem 3.2.8. *Computing intermediate stacks using Algorithm 3.4 has a bit complexity of* $\tilde{O}(n^{10}(n + \tau)^2)$.

Proof. Both R and R' are of magnitude $(n^2, n(n + \tau))$. Thus, merging the roots (Step 3 of Algorithm 3.4 has a complexity of $\tilde{O}(n^{10}(n + \tau)^2)$ (Theorem 2.5.19). Picking the roots is just a combinatorial iteration over the result and requires no expensive operation.

The bitstream-Descartes instances can be bounded using Lemma 3.2.7. Since all $\beta_1, \ldots, \beta_{s-1}$ are roots of R', we have

$$\sum_{i=1}^{n-1} \log \frac{1}{\operatorname{sep}(f(\beta_i, y))} = O(n^3(n + \tau)).$$

Therefore, we can proceed analogously to the analysis of the fiber construction in the direct approach of the curve analysis and obtain $\tilde{O}(n^{10}(n + \tau)^2)$ again.

Finally, we have to consider the leftmost and rightmost intermediate stacks. We restrict ourselves to the leftmost one; the rightmost is analogous. The magnitude of $f(\beta_0, y) \in \mathbb{Z}[y]$ is $(n, n^2(n+\tau))$, since β_0 has a bitsize of $\sigma = O(n(n+\tau))$. Therefore, the Descartes method has a bit complexity of $\tilde{O}(n^8(n + \tau)^2)$. $\square$

In Algorithm 3.3 for the (direct) curve analysis, steps 10 to 12 can be replaced by Algorithm 3.4 – the number of points in the intermediate fibers is determined in both variants. Furthermore, computing the intermediate stacks reveals more geometric information, since a sample point in the interior of each x-monotone segments is computed as well.

3.2.3. The approach for non-generic curves

We assume that the direct analysis (Algorithm 3.3) reported a failure indicator because the curve has a vertically asymptotic arc, or the m-k-bitstream-Descartes method failed for some critical x-coordinate α. Still, we can assume that the critical x-coordinates of the curve have been computed in the projection step, but we need to come up with the f-stacks using a different method.

Analysis in a sheared system:

We start by applying a shear transformation on the curve $V(f)$ (Definition 2.3.30) with shear factor s, to obtain $\mathrm{Sh}f_s$ and the curve $\mathrm{Sh}V(f_s)$. On the sheared curve, we apply the direct approach (Algorithm 3.3) to analyze the curve in a sheared coordinate system. If this steps fails, a new shear factor s is chosen, and the analysis is repeated.

We have to specify how to choose the shear factors. Our choice will show that the loop terminates after a constant number of chosen shear factors in expectancy. First, we argue that there are not too many "bad" shear factors, that is, values $s \in \mathbb{R}$ such that $\mathrm{Sh}_s f$ is not in generic position. We mention it does not suffice to exclude all directions that make a line joining two critical points of $V(f)$ vertical (such an argument was used in Lemma 2.3.31). This argument implies that critical points remain critical after applying a shear. However, this is only true for singular points. Non-singular critical points always become non-critical when shearing with a non-zero shear factor.

A geometric concept related to non-generic curves is the *double tangent*; indeed, if a line in $\mathbb{R}^2$ is tangential in at least two different points of the curve, the corresponding direction, chosen as the vertical direction, leads to a non-generic curve. A geometrical argument that only finitely many such directions exist can be found in [Ker06, Appendix A]. The proof uses dualizations of curves, and does not directly lead to an upper bound on the number of bad shear factors. Because we need such an upper bound, we use an algebraic proof [BPR06, Prop. 11.23].

Lemma 3.2.9. *Let f be square-free of total degree n, and let S denote a variable. Define*

$$\begin{aligned} F(S,x,y) &:= \mathrm{Sh}_S(f) = f(x+Sy,y) \\ D(S,x) &:= \mathrm{res}_y(F, \frac{\partial F}{\partial y}) \\ \Delta(S) &:= \min_k \left\{ \mathrm{sres}_k(D, \frac{\partial D}{\partial x}) \mid \mathrm{sres}_k(D, \frac{\partial D}{\partial x}) \neq 0 \right\}. \end{aligned}$$

If a curve $V(\mathrm{Sh}_s(f))$ (with $s \in \mathbb{R}$) is not in generic position, then s is a root of $\mathrm{lcf}_y(F) \in \mathbb{Z}[S]$ or of $\Delta \in \mathbb{Z}[S]$.

Proof. Instead of a complete formal proof that requires additional algebraic machinery (a proof that uses Puiseux series can be found in [BPR06, Prop. 11.23]), we offer an intuitive argument. Assume that $\mathrm{Sh}_s(f)$ is not in generic position and that $(\mathrm{lcf}_y(F))(s,x,y) \neq 0$ (this means that $\mathrm{Sh}_s(f)$ has cy^n with $c \in \mathbb{R}$ as its leading term, and thus has no vertical asymptotic segments). Note that $D(S,x)$ defines an algebraic curve and has the property that, for a given s, $D(s,x) \in \mathbb{R}[x]$ has as roots the critical x-coordinates of the curve $F(s,x,y) = \mathrm{Sh}_s(f)$ (by the specialization property). Since $\mathrm{Sh}_s(f)$ is not generic, the number of distinct (complex) critical x-coordinates decreases at s because two critical points become covertical (formally, showing this is the hardest part of the proof). But the number of distinct critical x-coordinates is determined by the index of the first non-vanishing subresultant coefficient of $D(s,x)$ and its derivative. Thus, if this quantity decreases, Δ must necessarily vanish. $\square$

We can now bound the maximal number of bad shear factors. The polynomial $\mathrm{lcf}_y(F)$ has up to n real roots. D is a bivariate polynomial of a degree of up to n^2 as a resultant of a polynomial of degree n. Δ is a subresultant coefficient of D, with respect to X, and of degree at most $(n^2)^2 = n^4$. Hence,

Corollary 3.2.10. *A curve f of degree n has at most $n^4 + n$ bad shear factors.*

Algorithmically, we proceed as follows: Integers from the range $\{1, \ldots, 2(n^4 + n)\}$ are chosen randomly as the shear factor until the analysis for the sheared curves succeeds. By our choice of the range, at least half of the shear factors in the range yield a generic curve. Thus, the expected number of tries until we find a successful shear factor is bounded by

$$\sum_{i=1}^{n^4+n} \frac{i}{2^i} < \sum_{i=1}^{\infty} \frac{i}{2^i} = 2.$$

It is not hard to see that $\mathrm{Sh}_s f$ is of magnitude $(n, \tau + \log n + n \log s)$. For our choice of s as above, the coefficient bitsize is thus $O(\tau + n \log n) = \tilde{O}(\tau + n)$. It follows that, in $\tilde{O}$-notation, the complexity of the direct curve analysis does not increase, and since we expect only 2 executions of the curve analysis until success, we have the following result.

Lemma 3.2.11. *Computing a curve analysis for some sheared curve $\mathrm{Sh}_s(f)$ with some shear factor $s \in \{1, \ldots, 2(n^4 + n)\}$ has an expected worst-case complexity of $\tilde{O}(n^{10}(n + \tau)^2)$.*

We note that this is already sufficient to prove that the topology computation for a curve $V(f)$ has the same complexity, since the curve analysis of a sheared version of $V(f)$ is sufficient to determine an isocomplex. However, to provide the curve analysis for the original curve, we need to do more.

Shearing back:

We assume from now on that $\mathrm{Sh}(f) := \mathrm{Sh}_s(f)$ has been analyzed. We exploit the information provided by that sheared analysis to construct the f-stacks of the original system. The main complication is that critical points of $V(f)$ do not need to be critical points of $V(\mathrm{Sh}(f))$ (and vice versa). Our strategy is to detect critical points of f in the sheared system. More precisely, we can restrict the search for critical points to a simpler-to-handle subclass by the following definition.

Definition 3.2.12 (event point). A point $p \in V(f)$ is an *event point* of $V(f)$ if its branch numbers are not equal to $(1, 1)$.

Note that an event point is always critical, but the converse is not true; for instance, a vertical cusp is critical but its branch numbers are $(1, 1)$.

From now on, we will frequently use the following notation. Points on $V(f)$ will be denoted by p or q, whereas points on the sheared curve $V(\mathrm{Sh}(f))$ are denoted by p^* and q^*. We also call $p^* = \mathrm{Sh}(p)$ the *sheared image* of p, or just the *shear* of p, and we call p in this situation the *sheared preimage* of p^*.

We first outline the workflow of the backshear process:

1. In the first step, we detect the sheared images $E^* := \{p_1^*, \ldots, p_t^*\}$ of the event points of $V(f)$. Note that $V(\mathrm{Sh}_s(f)) \setminus E^*$ induces a decomposition into a collection of

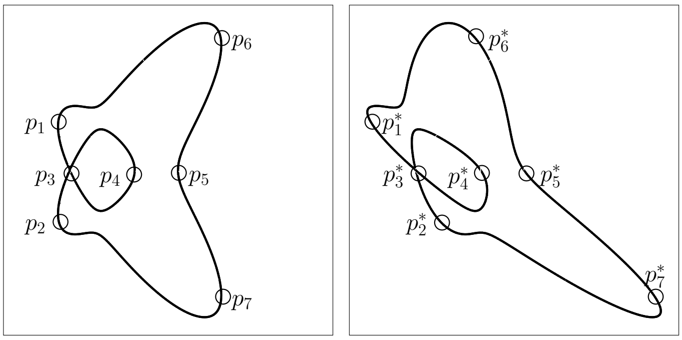


Figure 3.13. On the left: A (non-generic) curve and its event points.
On the right: Its sheared curve (with shear factor 2) and the sheared event points. Note that all p_i^* except p_3^* are non-critical points of the sheared curve.

(not x-monotone) segments $s_1^*, \ldots, s_u^*$ and that the corresponding preimage segments $s_1, \ldots, s_u$ are x-monotone segments of $V(f)$.

2. As the second step, we compute the endpoints of each s_i^*. This is simple, since we can exploit the curve analysis of the sheared curve.
3. In the third step, we compute the endpoints of each s_i. For that, it is enough to compute the preimage of each sheared event point and the (formal) preimage of each unbounded curve end of the sheared curve.
4. Knowing the endpoints of each s_i and the coordinates of each event point, we finally construct the f-stacks, using the oracle-bitstream-Descartes method (Section 2.6.2).

Critical points of $V(f)$ are defined as intersections of the curves $V(f)$ and $V(\frac{\partial f}{\partial y})$. Thus, their sheared images are the intersection points of the sheared curves $V(\mathrm{Sh}(f))$ and $V(\mathrm{Sh}(\frac{\partial f}{\partial y}))$. Therefore, to detect critical points in the sheared system, we can concentrate on the x-coordinates which are roots of

$$R_{\mathrm{ev}} := \mathrm{res}_y(\mathrm{Sh}(f), \mathrm{Sh}(\frac{\partial f}{\partial y})).$$

Note the subtle difference to the resultant

$$R_{\mathrm{sh}} := \mathrm{res}_y(\mathrm{Sh}(f), \frac{\partial(\mathrm{Sh}(f))}{\partial y})$$

whose roots correspond to the critical x-coordinates of the sheared curve.

We identify the event points as follows. We first create the f-stack of $V(\mathrm{Sh}(f))$ for any root of R_{ev} and of R_{sh}, and also intermediate stacks with respect to these f-stacks (Algorithm 3.4). Next, we iterate through all fiber points in all f-stacks over a root of R_{ev}; let p^* denote the current point, and p its sheared preimage. We want to determine whether p is an event point or not. Let (ℓ, r) denote the branch numbers of p^*. If $\ell + r \neq 2$, it follows that the branch numbers of p do not add up to 2 as well and thus, p is clearly an event point. Otherwise, if $\ell + r = 2$, there are precisely two segments adjacent to p^*. We use the intermediate stacks to find sample points q_1^* and q_2^* on both segments. In this situation, we have the following lemma.

Lemma 3.2.13. *The point p^* is the shear of an event point if and only if* $\operatorname{sign}((\mathrm{Sh}\frac{\partial f}{\partial y})(q_1^*)) \neq \operatorname{sign}((\mathrm{Sh}\frac{\partial f}{\partial y})(q_2^*))$.

Proof. Notice that $(\mathrm{Sh}\frac{\partial f}{\partial y})(q_i^*) = \frac{\partial f}{\partial y}(q_i)$. We let $q_i = (a_i, b_i)$ and observe that $\frac{\partial f}{\partial y}(q_i) = f(a_i, y)'(b_i)$. Hence it suffices to show in the original system that p is an event point if and only if $\operatorname{sign}(f(a_1, y)'(b_1)) \neq \operatorname{sign}(f(a_2, y)'(b_2))$. The plane decomposes into the curve $f = 0$ and into regions that are positive ($f > 0$) or negative ($f < 0$). Consider the x-monotone segments σ_i of f that connect p to q_i, $i = 1, 2$.

If σ_1, σ_2 extend to different sides of p, then p is not an event point, and $\sigma := \sigma_1 \cup \sigma_2$ is x-monotone and separates two regions. W.l.o.g. let the region below σ be negative. A vertical upward ray at $x = a_i$ leaves this region at the simple root b_i of $f(a_i, y)$, so the region above σ is positive, and $f(a_i, y)'(b_i) > 0$ for both i (Figure 3.14, left).

If σ_1, σ_2 extend to the same side of p, then p is an event point, and w.l.o.g., there is a negative region above σ_1 and below σ_2. An upward vertical ray at $x = a_1$ enters this negative region at the simple root b_1 of $f(a_1, y)$, hence $f(a_1, y)'(b_1) < 0$. A similar ray at $x = a_2$ leaves this negative region at b_2, hence $f(a_2, y)'(b_2) > 0$ (Figure 3.14, right). $\square$

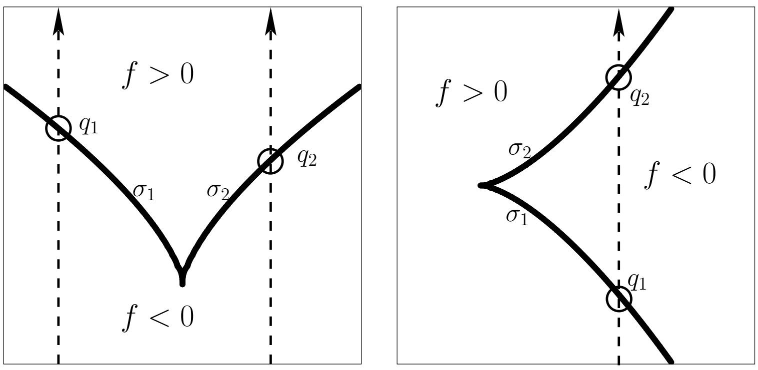


Figure 3.14. Illustration of the proof of Lemma 3.2.13.

Note that we can use this simple criterion only because we restricted the search to sheared event points. Detecting sheared critical points instead would be more involved.

The sign of $(\mathrm{Sh}\frac{\partial f}{\partial y})(q_i^*)$ can be computed using a two-dimensional analogon of Algorithm 2.18 (sign of $f(\alpha)$). For both the x- and y-coordinate of q_i^*, we know isolating intervals that can be refined up to any precision. Thus, we can compute arbitrarily small boxes that contain the point q_i^*, using interval arithmetic. For the evaluation of a bivariate polynomial $g = \sum_{i=0}^{n} a_i(x)y^i \in \mathbb{Z}[x][y]$ at a box $B := I_x \times I_y$ we define the evaluation function as follows:

$$\Box g(B) := \Box a_0(I_x) + I_y(\Box a_1(I_x) + I_y(\Box a_2(I_x) + \ldots (\Box a_{n-1}(I_x) + I_y \Box a_n(I_x)) \ldots).$$

The result is a real interval that contains the actual value $g(q_i^*)$. We evaluate $\Box\mathrm{Sh}\frac{\partial f}{\partial y}(B)$. If zero is not contained in the result, the sign is determined, otherwise, the box is refined (by refining x- and y-coordinates). Note that this eventually happens, since the box converges to q_i^*, consequently, the value $\Box\mathrm{Sh}\frac{\partial f}{\partial y}(B)$ converges to $\mathrm{Sh}\frac{\partial f}{\partial y}(q_i^*) \neq 0$.

After having identified the sheared event points, we construct a set of segments $s_1^*, \ldots, s_u^*$ on $V(\mathrm{Sh}(f))$, not necessarily x-monotone, whose endpoints are sheared event points or symbolic points at infinity. In fact, we are only interested in the endpoints of every segment here. It is a simple combinatorial task to construct these segments by a simple traversal of the segmentation of $V(\mathrm{Sh}(f))$ with respect to all roots of R_{ev} and R_{sh}. Let $s_i := \mathrm{Sh}_{-s}(s_i^*)$ denote the preimage of such a segment. By construction, s_i is an x-monotone segment of $V(f)$ (otherwise, it would have an event point in its interior). We aim for computing the endpoints of each s_i and thus, we have to explicitly compute the preimage of each sheared event point and each unbounded curve end. We will call $s_1, \ldots, s_u$ *event-bounded segments*, and likewise, $s_1^*, \ldots, s_u^*$ *sheared event-bounded segments* in the following description.

Let p^* denote a fixed sheared event point. As its sheared preimage p is an event point and thus critical, it is an element of a fiber of a critical x-coordinate of $V(f)$. We next determine in which fiber it is contained. For that, we use interval arithmetic. Approximate p^* by an axis-aligned box B (given by the isolating intervals for the x- and y-coordinates) and explicitly perform the transformation Sh_{-s} on B. This yields a parallelogram P_B, and its x-range covers at least one critical x-coordinate of $V(f)$. Retry with a refined box B until the x-range of P_B covers exactly one critical x-coordinate α. The event point p is then contained in the fiber of α, or in other words, the x-coordinate of p is α (see also Figure 3.15)

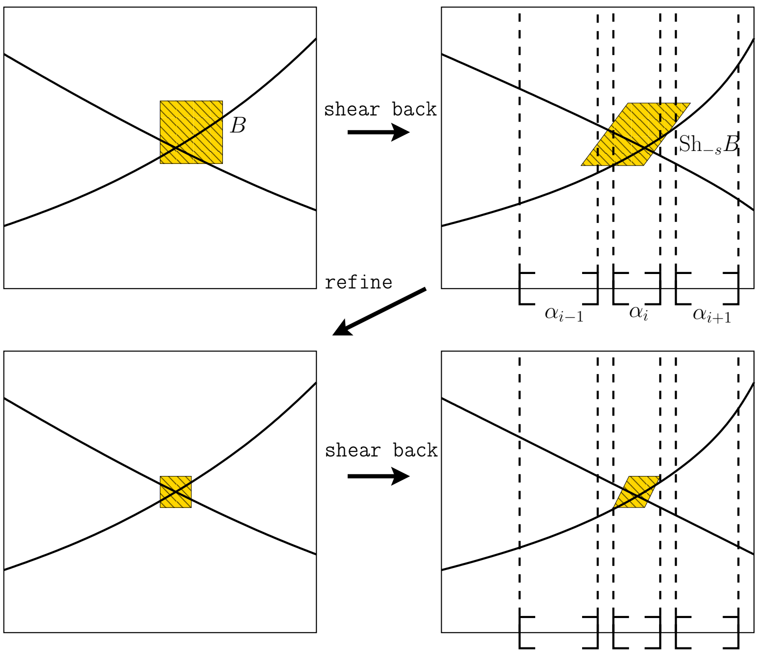


Figure 3.15. Above, the box of the event point contains three critical x-values in its x-range. After one more refinement, it only contains α_i, so the x-value of the event point must be α_i.

In fact, the explicit shear of the box B into P_B is only needed conceptually since the x-range of P_B can be read off immediately from the coordinates of B. Let (x_ℓ, y_ℓ) and

(x_r, y_r) denote the lower-left and upper-right corners of B. Then, the x-range of P_B is $[x_\ell + sy_\ell, x_r + sy_r]$ (assuming that $s \geq 0$). Thus, the algorithm simplifies to: Refine B until $[x_\ell + sy_\ell, x_r + sy_r]$ contains precisely one critical x-coordinate of $V(f)$.

After the critical x-coordinate of the event point has been computed, we already know isolating and refineable intervals for its x- and y-coordinates. For the latter, we can simply use the y-interval of the sheared image, because the y-coordinate is invariant under the shear transformation.

The next task is to compute the preimage of unbounded curve ends. What do we mean by this? Let s^* be a sheared event-bounded segment with a (symbolic) infinite endpoint p^*. Since we excluded vertically asymptotic arcs in the sheared curve, either $p^* = -\infty$ or $p^* = \infty$, which means that the segment is unbounded in the x-direction. When we shear back the segment s^*, its preimage s is unbounded as well, and has a formal endpoint. Thus, it makes sense to talk about the preimage p of p^*.

We exploit the fact that a vertically asymptotic segment with endpoint $(\alpha, \pm\infty)$ is only possible if α is a root of the leading coefficient $\mathrm{lcf}_y(f)$ of f, considered as a polynomial in y. So, let $\varphi_1, \ldots, \varphi_t$ denote the real roots of $\mathrm{lcf}_y(f)$ (they form a subset of the critical x-coordinates). Our goal is to construct $2t + 2$ unbounded regions, called *buckets*, that isolate the formal endpoints $+\infty, -\infty$, and $(\varphi_i, \pm\infty)$ from each other for $i = 1, \ldots, t$.

These buckets are constructed as follows. We compute rational intermediate values that separate the φ_i's from each other. For technical reasons, we need two intermediate values between φ_i and φ_{i+1} instead of a single one. This requires a simple adaption of Algorithm 2.14 (to be precise, replace Step 3 of the algorithm by $q_{0,1} \leftarrow -2^{\tau+2}, q_{0,2} \leftarrow -2^{\tau+1}$, $q_{m,1} \leftarrow 2^{\tau+1}, q_{m,2} \leftarrow 2^{\tau+2}$, and replace Step 7 by $q_{i,1} \leftarrow \frac{3c_i+d_i}{4}, q_{i,2} \leftarrow \frac{c_i+3d_i}{4}$). Let $q_{0,1}, q_{0,2}, \ldots, q_{n,1}, q_{n,2}$ be the result.

We define a box B containing all fiber points over any of those intermediate values (Figure 3.16). Observe the following properties. The vertical lines $x = q_{i,j}$ decompose

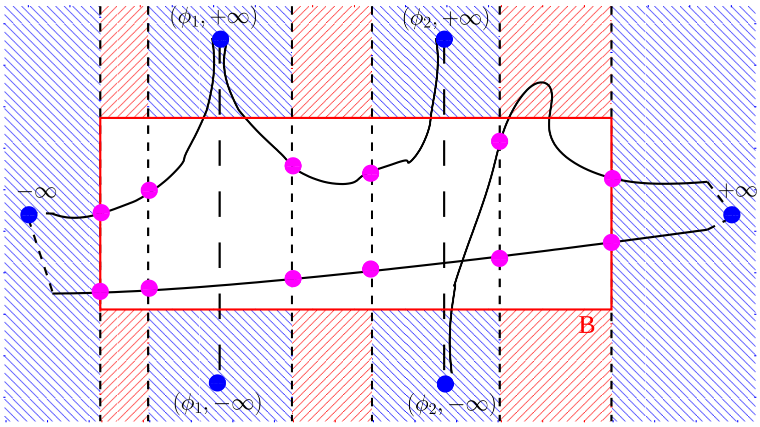


Figure 3.16. The box B, and the vertical lines at the intermediate positions (dotted lines) decompose the region outside B into full buckets (shaded blue) and empty buckets (shaded red).

$\mathbb{R}^2 \setminus B$ into several unbounded regions (buckets), one that is unbounded on the left-hand side, one that is unbounded on the right-hand side, and several vertically unbounded regions. Moreover, the formal endpoints $-\infty, +\infty, (\varphi_i, \pm\infty)$ are "strongly" isolated by these buckets, which means that each of them is contained in a different bucket, and for two buckets containing such formal endpoints, there is always an empty bucket in between (because we have chosen two intermediate values). We also call buckets that contain a formal endpoint *full buckets*, in contrast to the *empty buckets* in between (Figure 3.16). Observe also that no segment of $V(f)$ ever changes from one bucket to another without passing through B. This is simply by construction, as a change of buckets would cause a fiber point over some intermediate value $q_{i,j}$ outside B, which is impossible.

How does this lead to a method for finding the formal endpoints of unbounded segments? We pick x-coordinates x_ℓ, x_r in the sheared coordinate system, such that the interval $[x_\ell, x_r]$ covers all critical x-coordinates of the sheared curve and the whole x-range of the sheared box $\mathrm{Sh}(B)$, with B as defined above. We compute the fiber points at x_ℓ, and x_r of $\mathrm{Sh}V(f)$ by isolating the real roots of $\mathrm{Sh}(f)(x_\ell, y)$ and $\mathrm{Sh}(f)(x_r, y)$, letting $q_1^*, \ldots, q_m^*$ denote all the fiber points. Each q_i^* represents an infinite endpoint of an unbounded segment, and since $\mathrm{Sh}(f)$ has no vertically asymptotic arc, each infinite endpoint is covered by some q_i^*. Each q_i^* is sheared back, with the same method as the event points, and it is determined in which buckets the preimage q_i can be contained. As soon as only one full bucket is left as an option, we set the endpoint of the corresponding segment of $V(f)$ to the unique formal endpoint that is contained in this full bucket. Fugure 3.17 depicts an example.

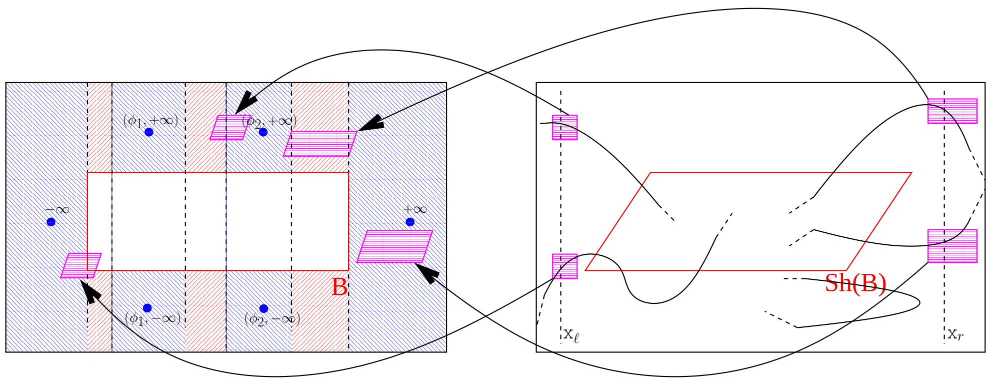


Figure 3.17. On the stack of $\mathrm{Sh}(f)$ at x_ℓ, the lower point has the preimage $-\infty$, since its backsheared box only touches the leftmost full bucket. Likewise, the upper point at x_ℓ has preimage $(\varphi_2, +\infty)$. For x_r, the lower point has preimage $+\infty$. The backsheared box of the upper point still overlaps with two full buckets, thus, further refinements are needed to deduce its preimage.

Why is this correct? Note that each q_i^* is outside the sheared box, thus, its preimage is outside B. Moreover, the whole infinite part of the sheared segment, starting at q_i^*, is outside the sheared box. Thus, the infinite part of the unsheared segment cannot enter the box anymore and is thus dedicated to the bucket in which q_i is contained. Its behavior at infinity is determined by the bucket of q_i. Clearly, this bucket must be a full bucket, since the segment must converge to one of the formal endpoints.

We remark that the use of empty buckets is only required for the complexity bound – in fact, the same algorithm also works when only choosing one intermediate value, which leads to full buckets only, which are adjacent to each other.

With the previous two operations on (sheared) event points and on unbounded segments, we know the endpoints of all event-bounded segments of $V(f)$. This information is sufficient to compute the f-stacks, as we show next. First of all, note that the vertical asymptote numbers $(\ell_-, \ell_+, r_-, r_+)$ are easily deducible by counting how many event-bounded segments have $(\alpha_i, -\infty)$ as their right endpoint, how many have $(\alpha_i, +\infty)$ as their right endpoint, and so on.

The next step is to construct the f-stack at α_i. For this, we aim to apply the oracle-Bitstream-Descartes method (Section 2.6.2). Note its prerequisites. We need to know the total number of fiber points m at α_i. This is done simply by adding the number of event points at α_i (this counts endpoints of segments) and the number of event-bounded segments whose x-range has α_i in its interior (this count the number of segments "passing" the fiber). Furthermore, for a set C of distinguished roots, we need to know its number c, and an "oracle" $\mathfrak{o}_S$ to decide whether an interval contains a root of C or not. We set C to be the set of event points at α_i. Their number is known and moreover, since each event point's y-coordinate is refineable to arbitrary precision, one can decide for any interval whether it contains an event point. The last prerequisite is that all roots not in C have odd multiplicity.

Lemma 3.2.14. *Let (α, β) be a non-event point of $V(f)$. Then, β is a root of $f(\alpha, y)$ with odd multiplicity.*

Proof. By using the same argument as in the proof of Lemma 3.2.13, the function $f(\alpha, y)$ changes its sign at the root β (Figure 3.14 (left)). □

After all, we can apply the oracle-Bitstream-Descartes method to isolate the real roots at α_i. The last data to assign are the branch numbers for each point in the fiber. For non-event points, they are $(1, 1)$ by definition. For event points, we count the number of event-bounded segments that have the corresponding event point as its left or right endpoint.

Algorithm 3.6 wraps up the backshear phase. We remark that the backshear routine can also be interpreted as follows. Given a curve analysis of $V(f)$ and some $s \in \mathbb{Z}$, compute the curve analysis of $\mathrm{Sh}_s(V(f))$. Our concrete backshear transformation would then be an application of this algorithm for the parameters $(\mathrm{Sh}_s(f), -s)$. This is important for practical reasons, since this transformation is faster than a completely new analysis from scratch, and it cannot fail.

We turn to the complexity analysis of Algorithm 3.5. First of all, we need a lemma on bivariate interval arithmetic that generalizes the univariate case in Lemma 2.5.20.

Lemma 3.2.15. *Let $g = \sum_{i=0}^{q} a_i(x) y^i \in \mathbb{Z}[x][y]$ with each $a_i \in \mathbb{Z}[x]$ of magnitude (p, τ). Let $B = I_x \times I_y$ be an axis-aligned box, with both $\mathrm{w}(I_{x,y}) < \varepsilon < 1$, containing a point (α, β). For each $y_0 \in \square g(B)$, we have*

$$|y_0 - g(\alpha, \beta)| \le \varepsilon 2^\tau 2^{p+q+3} \max\{1, |\alpha|\}^{p-1} \max\{1, |\beta|\}^q.$$

Proof. We do an induction on q. For $q = 0$, the statement reduces to the univariate case and follows from Lemma 2.5.20. Assume that the statement holds for polynomials of degree $q - 1$. We write

$$g(\alpha, \beta) = a_0(\alpha) + \beta \tilde{g}(\alpha, \beta)$$

Algorithm 3.5. Backshear process

INPUT: $f \in \mathbb{Z}[x,y]$ with critical x-coordinates $\alpha_1, \ldots, \alpha_s$, $s > 0$, and a curve analysis for $\mathrm{Sh}(f) := \mathrm{Sh}_s(f)$.
OUTPUT: f-stacks of $V(f)$ for each α_i

procedure BACKSHEAR($f, s, \mathrm{Sh}(f)$)
  $R_{\mathrm{sh}} \leftarrow$square-free part of $\mathrm{res}_y(\mathrm{Sh}(f), \frac{\partial \mathrm{Sh}(f)}{\partial y})$
  $R_{\mathrm{ev}} \leftarrow$square-free part of $\mathrm{res}_y(\mathrm{Sh}(f), \mathrm{Sh}(\frac{\partial f}{\partial y}))$
  Call MERGE_ROOTS($R_{\mathrm{sh}}, R_{\mathrm{ev}}$).
  Construct intermediate stacks with respect to the roots of $R_{\mathrm{sh}} \cdot R_{\mathrm{ev}}$
  Compute $\mathrm{sign}(\mathrm{Sh}\frac{\partial f}{\partial y}(q^*))$ for any intermediate fiber point q^*.
  Identify sheared event points$\rightsquigarrow E^*$
  Determine the sheared preimage of each element of E^*
  Compute stacks at intermediate values with respect to the roots of $\mathrm{lcf}_y(f)$ and construct a box B containing all their fiber points.
  Compute the fiber of $\mathrm{Sh}(f)$ at x-coordinates x_ℓ and x_r at the left (right) of the x-range of $\mathrm{Sh}(B)$, and of each critical x-coordinate of $\mathrm{Sh}(f) \rightsquigarrow U^*$
  Determine the bucket of each sheared preimage in U^* to determine the formal endpoint of unbounded segments of f
  Construct the list of event-bounded segments of f
  Determine the number of vertical asymptotes, and the number of fiber points of each f-stack
  Apply the ORACLE_DESCARTES on each $f(\alpha_i, y)$
  Assign branch numbers to each fiber point
end procedure

with $\tilde{g} = \sum_{i=0}^{q-1} a_{i+1}(x)y^i$, and

$$y_0 = \tilde{a_0} + (\beta + \nu)\tilde{y_0}$$

with $\tilde{a_0} \in \square a_0(I_x)$, $\nu \in [-\varepsilon, \varepsilon]$, and $\tilde{y_0} \in \square \tilde{g}(B)$. Then, it follows that

$$\begin{aligned}
& |y_0 - g(\alpha, \beta)| \\
\leq \; & |a_0(\alpha) - \tilde{a_0}| + |\nu \tilde{y_0} + \beta(\tilde{y_0} - \tilde{g}(\alpha, \beta))| \\
\overset{IA}{\leq} \; & \varepsilon 2^\tau 2^p \max\{1, |\alpha|\}^{p-1} + \varepsilon|\tilde{y_0}| + \beta\varepsilon 2^\tau 2^{p+q+2} \max\{1, |\alpha|\}^{p-1} \max\{1, |\beta|\}^{q-1}.
\end{aligned}$$

Note that with $\tilde{a_i} \in \square a_i(I_x)$,

$$\begin{aligned}
|\tilde{y_0}| \;\; & \leq \;\; |\tilde{a_1}| + (|\beta| + |\nu_1|)(|\tilde{a_2}| + (|\beta| + |\nu_2|)(|\tilde{a_3}| + \ldots (|\tilde{a}_{q-1}| + \beta + |\nu_q|\tilde{a_q}) \ldots) \\
& \leq \;\; (\max_{i=1,\ldots,q} |\tilde{a_i}|) \sum_{i=0}^{q-1} (|\beta| + \varepsilon)^i \\
& \leq \;\; (\max_{i=1,\ldots,q} |\tilde{a_i}|) \sum_{i=0}^{q-1} (2\max\{1, |\beta|\})^i \\
& \leq \;\; (\max_{i=1,\ldots,q} |\tilde{a_i}|) 2^q \max\{1, |\beta|\}^{q-1}
\end{aligned}$$

and $\max_{i=1,\dots,q}|\tilde{a_i}| \leq 2^\tau 2^{p+1}\max\{1,|\alpha|\}^p$ by a similar argument. Thus,

$$\begin{aligned}|y_0 - g(\alpha,\beta)| &\leq \varepsilon 2^\tau 2^p \max\{1,|\alpha|\}^{p-1} + \varepsilon|\tilde{y}| + \beta\varepsilon 2^\tau 2^{p+q+2}\max\{1,|\alpha|\}^{p-1}\max\{1,|\beta|\}^{q-1} \\ &\leq \varepsilon 2^\tau \max\{1,|\alpha|\}^{p-1}\max\{1,|\beta|\}^q \underbrace{(2^p + 2^{p+1+q} + 2^{p+q+2})}_{\leq 2^{p+q+3}}. \quad \square\end{aligned}$$

Corollary 3.2.16. *In the situation of Lemma 3.2.15, if*

$$\varepsilon < m_{\alpha,\beta} := \frac{|g(\alpha,\beta)|}{2^{p+q+4}2^\tau \max\{1,|\alpha|\}^{p-1}\max\{1,|\beta|\}^q},$$

then, $0 \notin \Box g(B)$.

Proof. If $\varepsilon < m_{\alpha,\beta}$, it follows from Lemma 3.2.15 that for each $y_0 \in \Box g(B)$

$$|y_0 - g(\alpha,\beta)| \leq \frac{|g(\alpha,\beta)|}{2},$$

thus, $0 \notin \Box g(B)$. $\square$

We start with the complexity analysis of the backshear algorithm. We will eventually show that its worst-case bit complexity is bounded by

$$\tilde{O}(n^{10}(n+\tau)^2),$$

which we will prove by proceeding step by step through Algorithm 3.5. The first lines (2-5) are similar to the direct method. Since both R_{ev} and R_{sh} are of magnitude $(n^2, n(n+\tau))$, all these steps are bounded by $\tilde{O}(n^{10}(n+\tau)^2)$. The first hard part is step 6, where we need to determine the sign of $\mathrm{Sh}(\frac{\partial f}{\partial y})$ for any point $(\alpha_i,\beta_{i,j})$, where $\alpha_1,\dots,\alpha_s$ are the chosen intermediate values and $\beta_{i,1},\beta_{i,m_i}$ are the fiber points of $\mathrm{Sh}(f)(\alpha_i,y)$. By Corollary 3.2.16, this requires refining the box of $(\alpha_i,\beta_{i,j})$ to a minimal side length of

$$m_{\alpha_i,\beta_{i,j}} := \frac{|\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha_i,\beta_{i,j})|}{2^{2n+4}2^\tau \max\{1,|\alpha_i|\}^{n-1}\max\{1,|\beta_{i,j}|\}^n}.$$

This means that the x- and y-coordinates of $(\alpha_i,\beta_{i,j})$ must be refined to $m_{\alpha_i,\beta_{i,j}}$. Note that all α_i are roots of the polynomial $R' := (R_{\mathrm{ev}}\cdot R_{\mathrm{sh}})'$ and that this polynomial is of magnitude $O(n^2, n(n+\tau))$.

Lemma 3.2.17.

$$\sum_{\alpha\in V(R')}\sum_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))}\log\frac{1}{m_{\alpha,\beta}} = O(n^3(n+\tau))$$

Proof.

$$\begin{aligned}\sum_{\alpha\in V(R')}\sum_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))}\log\frac{1}{m_{\alpha,\beta}} &\leq \sum_{\alpha\in V(R')}\sum_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))}\log\frac{2^{2n+4}2^\tau\max\{1,|\alpha|\}^n\max\{1,|\beta|\}^n}{\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha,\beta)} \\ \leq\ O(n^3(n+\tau)) + \sum_\alpha\sum_\beta n\log\max\{1,|\alpha|\} &+ \sum_\alpha\sum_\beta n\log\max\{1,|\beta|\} + \sum_\alpha\sum_\beta\frac{1}{\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha,\beta)}\end{aligned}$$

We show that each of the summands is bounded by $O(n^3(n+\tau))$. For the first summand

$$\begin{aligned}
& \sum_{\alpha\in V(R')}\sum_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))} n\log\max\{1,|\alpha|\} \\
\leq\ & n^2 \sum_{\alpha\in V(R')} \log\max\{1,|\alpha|\} \\
\leq\ & n^2 \log \mathrm{Mea}(R') = O(n^3(n+\tau)).
\end{aligned}$$

In the second summand, we exploit the multiplicativity of the Mahler measure to obtain

$$\begin{aligned}
& \sum_{\alpha\in V(R')}\sum_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))} n\log\max\{1,|\beta|\} \leq n \sum_{\alpha\in V(R')} \log\frac{\mathrm{Mea}(\mathrm{Sh}(f)(\alpha,y))}{\mathrm{lcf}(\mathrm{Sh}(f)(\alpha,y))} \\
\leq\ & n\log\frac{\mathrm{Mea}\prod_{\alpha\in V(R')}\mathrm{Sh}(f)(\alpha,y)}{\prod_{\alpha\in V(R')}\mathrm{lcf}(\mathrm{Sh}(f)(\alpha,y))} \leq n\log\frac{\mathrm{Mea}\frac{\mathrm{res}_x(\mathrm{Sh}(f),R')}{\mathrm{lcf}(R')^n}}{\frac{\mathrm{res}(\mathrm{lcf}_y(\mathrm{Sh}(f)),R')}{\mathrm{lcf}(R')^n}} \\
\leq\ & n\log\mathrm{Mea}(\mathrm{res}_x(\mathrm{Sh}(f),R')) + n\log\mathrm{lcf}(R')^n.
\end{aligned}$$

And by the same argument as in Lemma 3.2.7, $\log\mathrm{Mea}(\mathrm{res}_x(\mathrm{Sh}(f),R')) = O(n^2(n+\tau))$.

Finally, we need to bound the last summand:

$$\begin{aligned}
& \sum_{\alpha\in V(R')}\sum_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))} \log\frac{1}{\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha,\beta)} = \sum_{\alpha\in V(R')}\log\prod_{\beta\in V(\mathrm{Sh}(f)(\alpha,y))}\frac{1}{\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha,y)} \\
=\ & \sum_{\alpha\in V(R')}\log\frac{\mathrm{lcf}(\mathrm{Sh}(f)(\alpha,y))^n}{\mathrm{res}(\mathrm{Sh}(f)(\alpha,y),\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha,y))} \\
=\ & n\log\prod_{\alpha\in V(R')}\mathrm{lcf}(\mathrm{Sh}(f)(\alpha,y)) - \log\prod_{\alpha\in V(R')}\mathrm{res}(\mathrm{Sh}(f)(\alpha,y),\mathrm{Sh}(\frac{\partial f}{\partial y})(\alpha,y)) \\
=\ & n\frac{\mathrm{res}(\mathrm{lcf}_y(\mathrm{Sh}(f)),R')}{\mathrm{lcf}(R')^n} - \log\prod_{\alpha\in V(R')} R_{\mathrm{ev}}(\alpha) \\
\leq\ & n\cdot\mathrm{res}(\mathrm{lcf}_y(\mathrm{Sh}(f)),R') - \log\frac{\mathrm{res}(R_{\mathrm{ev}},R')}{\mathrm{lcf}(R')^{n^2}} \\
\leq\ & n\cdot\mathrm{res}(\mathrm{lcf}_y(\mathrm{Sh}(f)),R') + n^2\log\mathrm{lcf}(R') = O(n^3(n+\tau)). \quad \square
\end{aligned}$$

We use this lemma to bound the cost of Step 6 of Algorithm 3.5. We can assume that, within $\tilde{O}(n^{10}(n+\tau)^2)$ bit operations, the x-coordinates of each root of R' are refined to an interval width of $O(n^4(n+\tau))$ (strong root isolation, Theorem 2.5.16); this suffices to make all x-coordinates small. It remains to consider the y-coordinates. For each $\alpha\in V(R')$, set $m_\alpha := \min\{m_{\alpha,\beta} \mid \beta\in V(\mathrm{Sh}(f)(\alpha,y))$. If all intervals of the fiber of $\mathrm{Sh}(f)$ over α are refined to a width of m_α, then clearly all signs can be determined. By Theorem 2.6.21 (Refinement of isolating intervals for bitstream polynomials), this requires

$$\tilde{O}(n^3(\tau_\alpha + \log\frac{1}{\mathrm{sep}(\mathrm{Sh}(f)(\alpha,y))} + \log\frac{1}{m_\alpha}))^2$$

bit operations, where 2^{τ_α} is an upper bound on the coefficients of $\mathrm{Sh}(f)(\alpha,y)$. Similar to the direct case of the curve analysis, $\sum_\alpha \tau_\alpha = O(n^2(n+\tau))$, $\sum_\alpha \log\frac{1}{\mathrm{sep}(\mathrm{Sh}(f)(\alpha,y))} =$

$O(n^3(n+\tau))$ by Lemma 3.2.7, and $\sum_\alpha \log \frac{1}{m_\alpha} = O(n^3(n+\tau))$ by Lemma 3.2.17. Thus, we arrive at $\tilde{O}(n^9(n+\tau)^2)$ bit operations when considering all critical x-coordinates.

We continue with the next steps of Algorithm 3.5. Identifying the sheared event points based on the computed signs works completely combinatorially using Lemma 3.2.13. The next step is to compute the x-coordinate of each preimage of a sheared event point.

Lemma 3.2.18. *Let p^* be a sheared event point with isolating box $B = I_x \times I_y$. Let R be the resultant of f and $\frac{\partial f}{\partial y}$. The sheared preimage p of p^* has x-coordinate α, which is a root of R. If $\mathrm{w}(I_x) < \frac{\mathrm{sep}(R)}{2}$ and $\mathrm{w}(I_y) < \frac{\mathrm{sep}(R)}{2s}$, then the x-range of $\mathrm{Sh}_s(B)$ is an isolating interval for α and R.*

Proof. The x-coordinate of p is a root of R since event points are critical points. Let $I_x = (x_\ell, x_r)$ and $I_y = (y_\ell, y_r)$. The x-range of $\mathrm{Sh}_s(B)$ is given by $(x_\ell + sy_\ell, x_r + sy_r)$ and thus, the width is $x_r - x_\ell + s(y_r - y_\ell) = \mathrm{w}(I_x) + s\mathrm{w}(I_y) < \mathrm{sep}(r)$. Since α is contained in the x-range, no other root of r is contained; hence, the x-range is an isolating interval. $\square$

Since $\log \frac{1}{\mathrm{sep}(R)} = O(n^3(n+\tau))$ and $\log s = O(\log n)$, we need for each sheared event point $O(n^3(n+\tau))$ bits of precision in its x-coordinate, and $O(n^3 \log n(n+\tau)) = \tilde{O}(n^3(n+\tau))$ bits for the y-coordinate. Again, applying Theorem 2.5.16 (strong root isolation), even $O(n^4(n+\tau))$ bits of approximation can be obtained for each root of R_{ev} in $\tilde{O}(n^{10}(n+\tau)^2)$. For the y-coordinates, we need $\tilde{O}(n^3(n+\tau))$ bits per (sheared) event point. We apply Theorem 2.6.22. The isolation of all fibers at roots of R_{ev} has already been bounded. It remains to further isolate up to n^2 sheared event points, which corresponds to the second terms of the bound in Theorem 2.6.22, and gives a bound of $\tilde{O}(n^{10}(n+\tau)^2)$ bit operations.

We next look at Step 9, where a box is computed containing all fiber points of intermediate stacks with respect to $\mathrm{lcf}_y(f)$. Note that computing the intermediate x-coordinates requires only $O(n^4\tau^2)$ bit operations, since $\mathrm{lcf}_y(f)$ is of magnitude (n, τ) and each intermediate value is of bitsize $O(n(\tau + \log n))$ (Theorem 2.5.17). Recall from the description of the method for unbounded segments that we need two intermediate stacks between two roots, but this does not affect the complexity. After all, there are up to $2n+2$ fibers to compute and each polynomial $f(\varphi, y)$ with φ some intermediate value has a magnitude of $(n, n^2(\tau + \log n))$. This yields a total complexity of $\tilde{O}(n^9\tau^2)$ to obtain all intermediate stacks. We have to bound the size of the obtained box B. The maximal and minimal points in a fiber are bounded in their bitsize by $O(n^2(\tau + \log n))$ (the coefficient size of $f(\varphi, y)$), thus, the whole box is bounded in the y-direction by $2^{cn^2(\tau+\log n)}$ with some constant c, and in the x-direction by $2^{c'n(\tau+\log n)}$. This implies that the x-range of the sheared box is of magnitude $O(n^2(\tau + \log n))$ and thus, x_ℓ and x_r chosen by the algorithm are also of this magnitude. To construct the fiber at these positions requires $\tilde{O}(n^{10}\tau^2)$ bit operations using the Descartes method, since $f(x_\ell, y)$ and $f(x_r, y)$ are both of magnitude $(n, n^3(\tau + \log n))$.

For each fiber point, we compute the (full) bucket in which its preimage is contained. In this step, we exploit the fact that we have chosen two intermediate values between consecutive critical x-coordinates. Since both intermediate values are of bitsize of at most $O(n(\tau + \log n))$ (Theorem 2.5.17), any empty bucket has a width of at least $\Omega(\frac{1}{n(\tau+\log n)})$. Therefore, we need, for each fiber point over x_ℓ or x_r, $O(n \log n(\tau + \log n)) = \tilde{O}(n\tau)$ bits for the y-coordinate. By Theorem 2.5.16 (strong root isolation), this is not more expensive than the root isolation itself, and therefore bounded by $\tilde{O}(n^{10}\tau^2)$.

Algorithm 3.6. Curve analysis

INPUT: $f \in \mathbb{Z}[x, y]$ square-free
OUTPUT: curve analysis of $V(f)$

procedure CURVE_ANALYSIS(f)
    Call CA_DIRECT(f). If successful, **return**
    while true **do**
        Choose $s \in \{1, \ldots, 2(n^4 + n)\}$ at random
        Call CA_DIRECT($\mathrm{Sh}_s(f)$)
        If successful, call BACKSHEAR($f, s, \mathrm{Sh}_s(f)$) and return
    end while
end procedure

Computing the event-bounded segments out of the information obtained so far is a simple combinatorial task and requires basically no arithmetic operations. As the next step, we apply the oracle-Bitstream-Descartes method on each fiber. Since the same complexity bound as for the m-k-bitstream-Descartes algorithm holds, this again requires $\tilde{O}(n^{10}(n + \tau)^2)$ bit operations. It also requires up to $O(n^4(n + \tau))$ bits per x-coordinate, which can be computed by the same number of bit operations. Finally, we have to take the cost of the "oracle steps" into account. To check whether an interval during the subdivision contains an event point, it is enough to refine the y-coordinate of each sheared event point to width $\mathrm{sep}(f(\alpha_i, y))$. Since $\log \frac{1}{\mathrm{sep}(f(\alpha_i,y))} = O(n^3(n + \tau))$, this requires $\tilde{O}(n^8(n + \tau)^2)$ bit operations per (sheared) event point (Theorem 2.6.22), and thus $\tilde{O}(n^{10}(n + \tau)^2)$ in total. The branch numbers can be assigned after a single iteration through the set of event-bounded segments.

In summary, we have seen that each step in the backshear process is bounded by $\tilde{O}(n^{10}(n + \tau)^2)$. We can finally combine all ingredients into a complete curve analysis algorithm. See Algorithm 3.6. Its expected worst-case complexity is bounded by $\tilde{O}(n^{10}(n+\tau)^2)$, as follows from Lemma 3.2.11 and by the complexity of the backshear procedure.

3.2.4. Computing additional information

We have so far concentrated on the minimal requirements for a curve analysis in order to provide the **Make_x_monotone** primitive. Several extension are possible if more information is required.

Curves with vertical components This is not really an extension, but a special case that was not handled before. We have assumed throughout the last section that f had no vertical line as a component. Let us now assume that f has a vertical line component. Then, f, considered as a polynomial in y, decomposes into a primitive part $\mathrm{pp}(f)$, and its content $\mathrm{cont}(f) \in \mathbb{Z}[x]$. The latter curve consists of the vertical components of f (Lemma 2.2.5); let $\varphi_1, \ldots, \varphi_s$ denote the real roots of $\mathrm{cont}(f)$. We apply the curve analysis algorithm to $\mathrm{pp}(f)$, but we extend the critical x-coordinates of it by $\varphi_1, \ldots, \varphi_s$, thus we create an f-stack at each position of a vertical line. After the curve analysis has been computed, we iterate through the φ_i's and set the `is_vertical` flag for each such f-stack. It is easily seen that the total complexity bound is not increased by these operations.

Horizontal asymptotes So far, unbounded segments have been grouped into the following types (compare page 26): segments with curve ends at $x = -\infty$ or $x = +\infty$ denoting segments that are unbounded in their x-coordinate, and segments with curve ends of type $x = (\alpha_i, -\infty)$ and $x = (\alpha_i, +\infty)$, where α_i is a critical x-coordinate (or, more precisely, a root of $\mathrm{lcf}_y(f)$). The latter segments are asymptotically converging towards the vertical line $x = \alpha_i$.

For the former type of segment, we can introduce a further distinction: segments unbounded in their x-coordinate can either be unbounded in their y-coordinate as well (then, they diverge to one of the four "corner points" $(\pm\infty, \pm\infty)$ of $\mathbb{R}^2$). Or, they have a horizontal asymptote $y = \beta_i$. Sometimes, it is useful to distinguish these cases. For instance, if the arrangement represents curves in a parametric space on a torus or cylinder, the lines $x = -\infty$ and $x = +\infty$ are glued together; for that reason, one needs additional information where the curve "hits" the lines at infinity (we give an application for this in Chapter 5).

Such information is computed via a simplified version of the approach to detect vertical asymptotes. Note that a horizontal line $y = \varphi$ can only be a horizontal asymptote for $V(f)$ if φ is a root of $\mathrm{lcf}_x(f)$. Thus, let $\varphi_1, \ldots, \varphi_t$ be roots of $\mathrm{lcf}_x(f)$. We compute, as in the vertical case, two intermediate values for each pair of consecutive roots (and additionally, two intermediate values between $-\infty$ and φ_1 and between φ_t and $+\infty$). This induces $2t+2$ horizontal stripes: two extremal unbounded stripes, and "empty" intermediate stripes alternating with stripes that contains a horizontal line $y = \varphi_j$. For each unbounded segment towards $x = \pm\infty$, it is enough to determine the stripe it is eventually contained in to determine its type.

For that, choose the rational value x_ℓ (x_r) on the left (right) of each critical x-coordinate, and on the left (right) of any intersection of $V(f)$ with one of the horizontal lines bounding one of the stripes just defined. In other words, in the fiber of $V(f)$ over x_ℓ (x_r), each point belongs to a unbounded segment with formal endpoint $x = -\infty$ ($x = +\infty$), and the segment cannot change its stripe anymore when running further towards its infinite endpoint. Thus, it is enough to determine the stripe in which each fiber point is contained in order to determine the type of the corresponding unbounded segment (Figure 3.18).

The complexity of this step is determined with the same methods as for the vertical case. The bitsize of each intermediate value is bounded by $O(n(\tau+\log n))$. Thus, all intersections with $V(f)$ are contained in an x-range whose boundaries are of bitsize $O(n^2(\tau + \log n))$. It follows that x_ℓ and x_r are of bitsize $O(n^2(\tau + \log n))$, and to determine the stripe of each fiber point, it is necessary to refine all roots to a precision ε such that $\log \frac{1}{\varepsilon} = O(n^2(\tau + \log n))$. This is because when refined to this precision, no isolating interval can cover more than one non-empty stripe. Using Theorem 2.5.16 (strong root isolation), this requires $\tilde{O}(n^8(n + \tau)^2)$ bit operations. The preceding computation of x_ℓ and x_r required up to $2n$ applications of Theorem 2.2.11, which requires $O(n\tau)$ arithmetic operations. This is clearly negligible.

Singular point detection The curve analysis has computed fibers for each critical x-coordinate of the curve, that is, for each x-coordinate where critical (and thus singular) points may arise. However, it does not give the definite answer to the question of whether a certain point is critical/singular or not. So far, we only have the following partial answers:

- If a point does not belong to a critical f-stack, it is not critical.

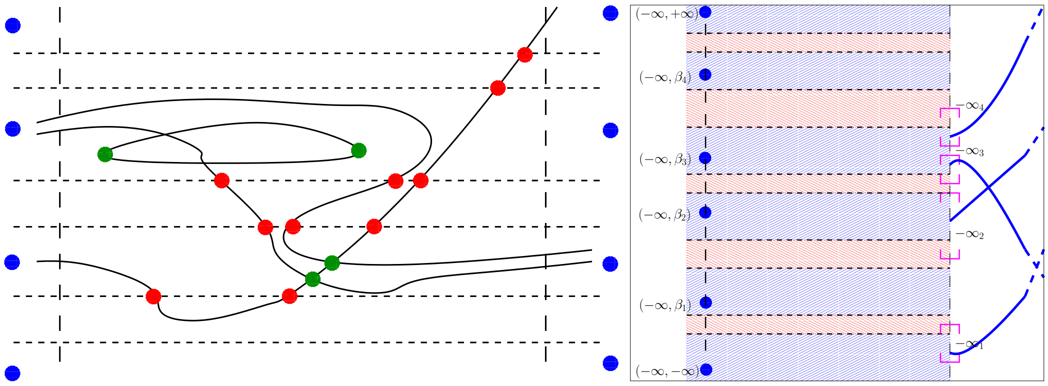


Figure 3.18. **On the left**: The dashed horizontal lines are chosen with respect to the roots of the leading coefficient $\mathrm{lcf}_x(f)$ (blue points). The extreme values x_ℓ and x_r (dashed vertical lines) are chosen on the left/right of any critical point of the curve (green points), and any intersection with an intermediate line (red points).
On the right: The isolating intervals in the fiber of x_ℓ. Each overlaps with a unique blue region, which determines the endpoint of the segment.

- If a point has branch numbers different from $(1,1)$, it is critical. If it has branch numbers that do not sum to 2, it is singular.
- If a point has a y-coordinate such that the m-k-bitstream-Descartes, or the oracle-bitstream-Descartes method identified it as a simple root, it is not critical.

The last item states that if the fiber has been constructed by the m-k-bitstream-Descartes method, all points except the distinguished one in the fiber (the one that might contain a multiple root) are non-critical and thus non-singular. Still, there is no guarantee that the distinguished root is really multiple. For instance, it might be the case that the critical x-coordinate is caused by a complex critical point with a real x-coordinate, thus, all real roots are simple and there is a pair of complex conjugate roots close to a real root such that the sign variation in the isolating interval of this root is increased.

Moreover, even if the root is multiple, there remains the question of whether it is singular or not. See Figure 3.19 for examples.

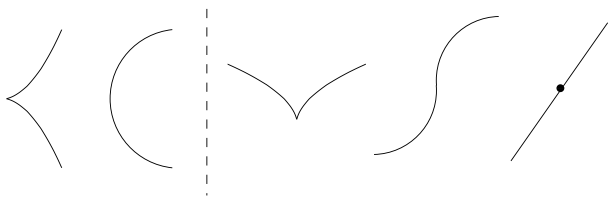


Figure 3.19. **On the left**: In both cases, the branch numbers are $(0,2)$, and the multiplicity of the isolating interval is at least 2. However, the left point is singular, whereas the right one is regular.
On the right: In all three cases, the branch numbers are $(1,1)$, and the multiplicity of the isolating interval is at least 3. But only the middle point among the three is regular.

We present an extension that allows setting a flag for each fiber point to denote whether a point is singular or not. The method will be based on the previous results of the analysis, but it also introduces more symbolic computations in the algorithm and might considerably worsen the practical performance (although the "partial answers" in the enumeration above already determine the flag in most instances).

Recall Lemma 2.3.16. If two polynomials g_1 and g_2 have exactly one common root of multiplicity k, then this root is given by

$$\beta = -\frac{\mathrm{coef}_{k-1}(\mathrm{Sres}_k(g_1, g_2))}{k \cdot \mathrm{coef}_k(\mathrm{Sres}_{k,k}(g_1, g_2))}.$$

We apply this result in the context of a polynomial $f(\alpha, y) \in \mathbb{R}[y]$, with α a critical x-coordinate. Let $k := \gcd(f(\alpha, y), f'(\alpha, y))$. We do not know whether there is exactly one multiple root of $f(\alpha, y)$, that is, exactly one common root of $f(\alpha, y)$ and $\frac{\partial f}{\partial y}(\alpha, y) = f'(\alpha, y)$. Let us set

$$\begin{aligned} p(x) &:= -\mathrm{coef}_{k-1}(\mathrm{Sres}_k(f, \frac{\partial f}{\partial y})) \\ q(x) &:= k \cdot \mathrm{coef}_k(\mathrm{Sres}_{k,k}(f, \frac{\partial f}{\partial y})). \end{aligned}$$

Note that $q(\alpha) \neq 0$, since $\mathrm{sres}_k(\alpha)$ is the first non-vanishing subresultant at α (Lemma 2.3.14). A simple consequence of Lemma 2.3.16 is the following:

Lemma 3.2.19. *If f is in generic position and α is a critical x-coordinate, then $(\alpha, \frac{p(\alpha)}{q(\alpha)})$ is the unique critical point in the fiber at α.*

Moreover, we can check whether $(\alpha, \frac{p(\alpha)}{q(\alpha)})$ is a critical point, and whether it is singular, as follows.

Lemma 3.2.20. *Let p, q be defined as above. For $g \in \mathbb{Z}[x, y]$, define*

$$H_g^{(k)}(x) := q(x)^{\deg_y(g)} g(x, \frac{p(x)}{q(x)}) \in \mathbb{Z}[x].$$

Then, for any critical x-coordinates, it holds that:

1. *If $H_f^{(k)}(\alpha) = H_{\frac{\partial f}{\partial y}}^{(k)}(\alpha) = 0$, then $(\alpha, \frac{p(\alpha)}{q(\alpha)})$ is a critical point of $V(f)$.*
2. *In this situation, $H_{\frac{\partial f}{\partial x}}(\alpha) = 0$, if and only if $(\alpha, \frac{p(\alpha)}{q(\alpha)})$ is a singular point of $V(f)$.*
3. *Otherwise, if $H_f^k(\alpha) \neq 0$ or $H_{\frac{\partial f}{\partial y}}(\alpha) \neq 0$, then $V(f)$ is not in generic position.*

Proof. This all follows simply by the fact that $H_g^{(k)}(\alpha) = 0$ if and only if $(\alpha, \frac{p(\alpha)}{q(\alpha)})$ is a point on $V(g)$. □

The directly preceding lemma proposes the following algorithm. We first adapt the direct method of the curve analysis algorithm. Assume that the fiber over α (a critical x-coordinate) has been computed using the m-k-bitstream-Descartes method. Then, all fiber points except the distinguished one are non-critical. We check whether $H_f^k(\alpha) = 0$ and whether $H_{\frac{\partial f}{\partial y}}^k(\alpha) = 0$. If not, the method fails (and a shear transformation is applied). If both vanish, there is a real critical point at α and thus, the distinguished fiber point must be critical. We additionally check whether $H_{\frac{\partial f}{\partial x}}(\alpha) = 0$ and set its singular flag depending on the result.

So far, we have only mentioned the direct case. If the curve analysis must be performed in a sheared coordinate system, the singular flags are set for the sheared curve. Slightly redefining the term "event point", all singular points are also labeled as event points. The backshear process is performed as before, and each event point of $V(f)$ inherits its singularity flag from its sheared image (note that being singular is a property that is invariant under shearing).

How costly are these steps? First of all, we have to bound the magnitude of $H_g^{(k)}$. Assume that g is of magnitude (n, τ), and p and q are of magnitude $(n^2, n(n+\tau))$. Then, it is not hard to see that the degree of $H_g^{(k)}$ is bounded by n^3. To bound the bitsize, note that $H_g^{(k)}$ can be written as the sum

$$H_g^{(k)} = \sum_{i,j} a_{i,j} x^i p(x)^j q(x)^{deg_y(f)-j}.$$

We consider $H_g^{(k)}$ as a polynomial in the formal coefficients $a_{i,j}$ and the coefficients of p and q. Note that, when fully expanded, $p(x)^j$ and $q(x)^{deg_y(f)-j}$ have up to n^{2n} summands. In total, this gives n^{4n+2} formal summands. Each coefficient is bounded in its bitsize by $O(n^2(n+\tau))$, and when adding up to n^{4n+2} such coefficients, this yields a maximal bitsize of $O(n^2(n+\tau) + n \log n) = O(n^2(n+\tau))$. Thus, $H_g^{(k)}$ is of magnitude $(n^3, n^2(n+\tau))$.

To check whether $H_g^{(k)}(\alpha) = 0$, we compute $\gcd(H_g^{(k)}, r)$, where r is the square-free part of $\mathrm{stha}_0(f)$, and evaluate the gcd at the boundaries of the isolating intervals of α (compare Algorithm 2.15). Computing this gcd requires up to

$$O(n^5 M(n^3 \cdot n^2(n+\tau))) = O(n^{10}(n+\tau))$$

bit operations (Theorem 2.4.19), and the resulting gcd is of magnitude $(n^2, n(n+\tau))$. The gcd computation has to be done at most n times (once for each $H_g^{(k)}$ with $k = 1, \ldots, n$), thus, all gcd computations are bounded by

$$\tilde{O}(n^{11}(n+\tau)) = \tilde{O}(n^{10}(n+\tau)^2).$$

For the evaluation of the gcd at the interval boundaries, recall that each boundary is of bitsize $O(n^3(n+\tau))$; thus, the evaluation of one point has a bit complexity of $\tilde{O}(n^7(n+\tau))$. Doing this for each critical x-coordinate results in a bit complexity of $\tilde{O}(n^9(n+\tau))$.

After all, we have shown that checking for singular values does not increase the total complexity. Also, the fact that the algorithm becomes slightly more restrictive (i.e., might reject more curves) does not affect the worst-case bound, since curves in generic position are still accepted by the algorithm. But again, we emphasize that the practical performance might well be affected by this optional step, because the whole operation is purely symbolic.

The method actually computes more information: when the (modified) direct analysis has been successful, it also verifies for each distinguished point that it is indeed critical (but it still does not decide whether the curve is in generic position). However, if a shear has been performed, this is no longer true, since being critical is not invariant under shearing. The simplest way to distinguish critical from non-critical points for general curves seems to be to apply a curve pair analysis to $V(f)$ and $V(\frac{\partial f}{\partial y})$ and to label the intersection points as critical points of $V(f)$. The details of the curve pair analysis will be the subject of the next section.

Summary

The curve analysis provides information about an algebraic curve at its critical fibers. This induces a decomposition of the curve into x-monotone segments and allows approximating any point of the curve to arbitrary precision. We have presented an algorithm that combines symbolic computations with faster numerical methods and have derived a worst-case bound that matches state-of-the-art solutions to this problem.

3.3. Algorithm for curve pair analysis

Let $f, g \in \mathbb{Z}[x, y]$ be fixed in this section, and assume that they are square-free and coprime. Recall the definition of a curve pair analysis (Definition 3.1.10) of $(V(f), V(g))$. For any critical x-coordinate of the curve pair (i.e., a critical x-coordinate for a single curve or a root of $\mathrm{res}(f, g)$) and any intermediate value between consecutive critical x-coordinates, we compute an fg-stack which captures the vertical ordering of the fiber points of $V(f)$ and $V(g)$ at the current position. Moreover, for each intersection point that is neither at a critical x-coordinate of $V(f)$ nor of $V(g)$, we have to determine the intersection multiplicity.

In what follows, we assume that the curve analyses of the single curves $V(f)$ and $V(g)$ have been computed and are available during the algorithm. The algorithm uses quite a similar method to that of the curve analysis. We identify three main phases, which are sketched next. Detailed descriptions follow in the subsequent sections.

Pre-stack phase First, the critical x-coordinates and suitable intermediate values are computed. The fg-stacks for x-coordinates without potential intersection point are easy to compute. For x-coordinates where intersections are possible, a structure called a *pre-stack* is computed which contains a preliminary ordering of the fiber points of f and g.

Definition 3.3.1 (pre-stack). For a curve pair $(V(f), V(g))$ and some $\alpha \in \mathbb{R}$, a pre-stack at α is a sequence of entries “f”, “g”, and “p”, that represents the vertical ordering of the fiber points of $V(f)$ and $V(g)$. “f” stands for a fiber point of $V(f)$, “g” for a fiber point of $V(g)$, and “p” stands for a *pair* of fiber points, one from $V(f)$ and one from $V(g)$. This pair can either stand for an intersection of $V(f)$ and $V(g)$ or it can represent a sequence “fg” or “gf” in the fiber. In the latter cases, we call the pair a *fake pair*.

Obviously, to produce the final fg-stack for α, we have to identify the fake pairs of the pre-stack, and replace them by appropriate subsequences (See also Figure 3.20).

Direct analysis An attempt is made to turn pre-stacks into valid fg-stacks by a “direct” analysis. Some exact information on each critical fiber is computed using symbolic computations. This is enough to identify intersections in the interior of segments of $V(f)$ and $V(g)$ and to deduce their intersection multiplicity. At the remaining fibers, we try to reduce the number of pairs in the pre-stack to exactly one remaining pair, called the *candidate pair*, and it is ultimately ensured that this candidate pair is indeed an intersection.

Similar to the direct approach for one curve, the approach described might fail on several occasions. If this happens, a sheared version of the curve pair must be used (see

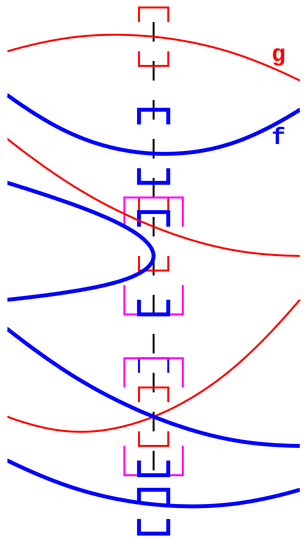


Figure 3.20. Illustration of a pre-stack. From bottom to top, it is defined by the string "fppfg". By looking at the picture, the fg-stack would be "fifgfg", thus, the first pair is an intersection point, but the second is a fake pair.

next paragraph). But at least the direct analysis will always work if the curve pair is in generic position.

Definition 3.3.2 (generic position of a curve pair). The curve pair $(V(f), V(g))$ is *in generic position* if $V(fg)$ is in generic position.

Analysis via a shear If the direct approach fails, a shear factor s is chosen at random, and the sheared curve pair $V(\mathrm{Sh}_s(f), \mathrm{Sh}_s(g))$ is analyzed by the direct method. In the event of failure, s is rechosen until the method succeeds.

If successful, all intersection points of $V(\mathrm{Sh}_s(f))$ and $V(\mathrm{Sh}_s(g))$ are approximable. By shearing back each intersection point, we can assign it to a unique pair in some pre-stack of the original curve pair. This reveals which pairs are intersections and which are not. The fake pairs can then be resolved (Figure 3.21).

There is one complication in this backshear process, namely that the multiplicity of intersection points in the interior of segments might remain unknown. In this case, one has to choose another shear factor to compute this multiplicity.

3.3.1. Details of the pre-stack computation

Projection phase The critical x-coordinates of the curve pair are given by the union of the roots of

$$r_f := \mathrm{res}_y(f, \frac{\partial f}{\partial y}) \quad r_g := \mathrm{res}_y(g, \frac{\partial g}{\partial y}) \quad r_{fg} := \mathrm{res}_y(f, g).$$

By two applications of Algorithm 2.16 (MERGE_ROOTS), we produce the sequence of critical x-coordinates and can determine for each of them whether they belong to r_f, r_g,

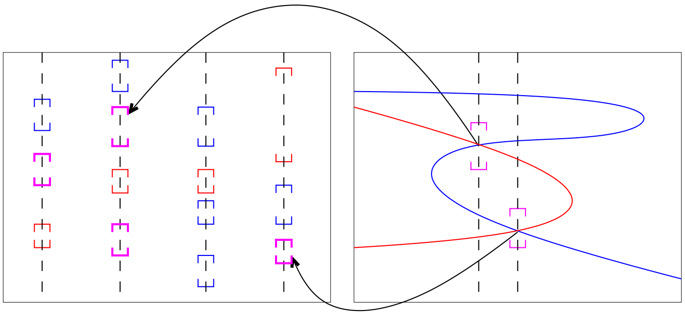


Figure 3.21. Illustration of the backshear process. On the left, we see pre-stacks of the original curve – pairs are drawn in pink (thicker). On the right, we see the sheared curves. Shearing back its two intersection points yields the pairs that contain intersection points – the other two pairs must be fake pairs.

or r_{fg}. We call critical x-coordinates which are not roots of r_{fg} *one-curve-critical*, roots of r_{fg} which are neither roots of r_f nor of r_g *curve-pair-critical*, and roots of both r_{fg} and of $r_f \cdot r_g$ *fully critical*. See also Figure 3.22.

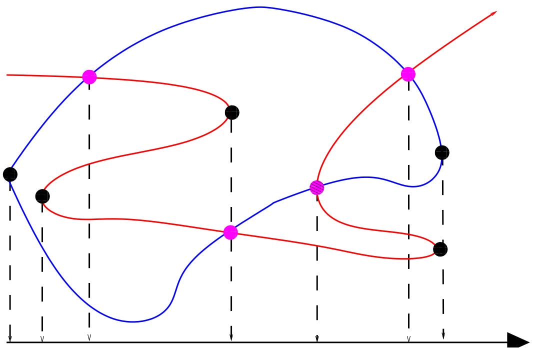


Figure 3.22. From left to right, the critical x-coordinates of the curve pair are: one-curve-critical, one-curve-critical, curve-pair-critical, fully critical, fully critical, curve-pair-critical, and one-curve-critical.

Intermediate values are computed similarly to the one-curve case. We isolate the real roots of $(r_f \cdot r_g \cdot r_{fg})'$ and pick intermediate values between consecutive roots (compare Section 3.2.2).

fg-stack creation without intersections For each α that is either one-curve-critical or an intermediate x-coordinate, fetch the f-stack and the g-stack at α (they are either computed already or can be obtained easily from the curve analysis by creating an additional intermediate stack). Refine the isolating intervals for the y-coordinates, both for $V(f)$ and $V(g)$, until they are all pairwise disjoint from each other. The order of the refined intervals then determines the fg-stack.

Algorithm 3.7. Pre-stack Computation

INPUT: $f, g \in \mathbb{Z}[x, y]$ square-free and coprime
OUTPUT: Critical x-coordinates and intermediate values for the curve pair $(V(f), V(g))$, fg-stacks for all intermediate values and one-curve-critical x-coordinates

procedure PRE_STACK(f, g)
  $r_f \leftarrow \mathrm{res}_y(f, \frac{\partial f}{\partial y}), r_g \leftarrow \mathrm{res}_y(g, \frac{\partial g}{\partial y}), r_{fg} \leftarrow \mathrm{res}_y(f, g)$
  Merge the roots of r_f, r_g and r_{fg} into one sequence $\alpha_1, \ldots, \alpha_s$
  Compute intermediate values $\gamma_0, \ldots, \gamma_s$
  For each γ_i, and each α_j that is not a root of r_{fg}, compute the fg-stack by refining the isolating intervals of the f-stacks and g-stacks to disjointness
  For each α_i that is root of r_{fg}, compute a pre-stack by refining the isolating intervals of the f-stacks and g-stacks until each interval overlaps with at most a single other
end procedure

Pre-stack computation For each α that is a root of r_{fg}, fetch the f-stack and the g-stack at α. Refine the isolating intervals for the y-coordinates until each interval for a fiber point of $V(f)$ overlaps with at most one interval for a fiber point of $V(g)$, and vice versa. This induces the pre-stack at α in a direct way: intervals for fiber points in $V(f)$ that are isolated from any interval of $V(g)$ are represented by "f", isolated intervals for fiber points of $V(g)$ are represented by "g", and pairs of isolating intervals are represented by "p". Note that if we knew which pairs were fake pairs, we could easily *split* them by further refining the involved intervals to disjointness, and we would be done. But we do not know yet which pairs are fake.

3.3.2. Details of the direct analysis

Symbolic precomputation For each curve-pair-critical value and each fully-critical value α we require that $\deg f(\alpha, y) = \deg_y f$ and that $\deg g(\alpha, y) = \deg_y g$, in other words, no real root of r_{fg} must be a root of $\mathrm{lcf}_y(f)\mathrm{lcf}_y(g)$. If this is not satisfied, the direct analysis fails (and we pass to phase three explained below). For each root α of r_{fg}, we compute $k_\alpha := \deg(\gcd(f(\alpha, y), g(\alpha, y)))$ using the principal subresultant coefficients of f and g (Lemma 2.3.14). We will write $k := k_\alpha$ for convenience, if it is clear which α is meant. Moreover, we compute $\mu_\alpha := \mathrm{mult}(\alpha, r_{fg})$, the multiplicity of α as a root of the resultant. This can be done using the square-free factorization of r_{fg} (Algorithm 2.8).

If $k_\alpha > 1$ for any curve-pair-critical x-coordinate α, the direct method fails, and we pass to phase three.

Splitting fake pairs Fix some curve-pair-critical x-coordinate α. We have ensured that $k_\alpha = 1$. It follows that there is exactly one intersection point of $V(f)$ and $V(g)$ at $x = \alpha$, and it must be real. Moreover, its intersection multiplicity is equal to μ_α (Theorem 2.3.33). Therefore, we can simply proceed as follows. We repeatedly loop through all pairs of the pre-stack and refine both intervals. If a pair is *split* in this step, that is, the intervals of the pair become disjoint, the entry "p" in the pre-stack is replaced by "fg" or "gf", depending on their order. This is done until all pairs except one have been split. The last remaining pair is relabeled as an intersection point and its intersection multiplicity is set to μ_α.

Alternatively, let α be some fully-critical value. The method just described cannot work here, since we cannot require that $k_\alpha = 1$ (even if the curve pair is in generic position). Therefore, we do not know the number of intersection points at α. We need to compute additional information to ensure termination in the presence of several intersection points. Consider a pair (I, J) of the pre-stack, consisting of an isolating interval I for a root of $f(\alpha, y)$ and an isolating interval J for a root of $g(\alpha, y)$. Recall from Section 2.6.4 that both I and J have a multiplicity m_I and m_J that yield an upper bound for the multiplicity of the contained root with respect to the corresponding polynomial. We set the multiplicity $m_{(I,J)}$ of the pair (I, J) to $m_{(I,J)} := \min\{m_I, m_J\}$.

Lemma 3.3.3. *Let $k := \gcd(f(\alpha, y), g(\alpha, y))$. If the curve pair is in generic position, there is always a pair at α with multiplicity k.*

Proof. If the curve pair is generic, it follows that there is only one intersection point at α, let β denote its y-coordinate. Then $\gcd(f(\alpha, y), g(\alpha, y)) = (y - \beta)^k$, thus, β is a root of multiplicity of at least k for both f and g. It follows directly that $m_{(I,J)} \geq k$. □

We repeatedly loop through all pairs of the pre-stack and refine both intervals. If a pair is *split* in this step, that is, the intervals of the pair become disjoint, the entry "p" in the pre-stack is replaced by "fg" or "gf", depending on their order. Otherwise, we update the multiplicity of the pair.

There are two termination conditions. First, if each pair in the pre-stack has a multiplicity of less than k, the direct method fails. Second, if only one pair remains in the pre-stack, this last pair is labeled as the *candidate pair*, and we proceed with the next α. Note that the latter condition will eventually happen if the curve pair is in generic position.

Checking the candidate pair Consider some fully-critical x-coordinate α and assume that (I, J) is the candidate pair. We know that there is at most one intersection, and if there is one, it must be represented by the candidate pair. But there is no guarantee that the candidate is indeed an intersection. A cheap argument to get a certified positive answer follows

Lemma 3.3.4. *If k_α is odd or μ_α is odd, then the candidate is an intersection point.*

Proof. If k_α is odd, the gcd of $f(\alpha, y)$ and $g(\alpha, y)$ has a real root, thus, there is a real intersection in the fiber. If $\mu_\alpha = \operatorname{mult}(\alpha, r_{fg})$ is odd, the intersection multiplicities sum to an odd number, according to Lemma 2.3.33. Since each pair of complex conjugate non-real intersection points must have the same intersection multiplicity (because of symmetry), there must be a real intersection as well. □

If both k_α and μ_α are even, we try to ensure the presence of an intersection symbolically, using the same method as for detecting singular points in the curve analysis (Section 3.2.4). Set $k := k_\alpha$ and define

$$\begin{aligned} p(x) &:= -\operatorname{coef}_{k-1}(\operatorname{Sres}_k(f, g)) \\ q(x) &:= k \cdot \operatorname{coef}_k(\operatorname{Sres}_{k,k}(f, g)) \end{aligned}$$

and

$$H_f^{(k)}(x) := q(x)^{\deg_y(f)} f(x, \frac{p(x)}{q(x)}) \in \mathbb{Z}[x]$$

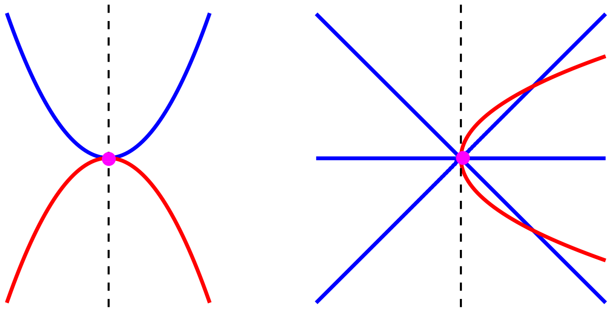


Figure 3.23. On the left, we see the curves $V(y-x^2)$ and $V(y+x^2)$. For $\alpha = 0$, $k_\alpha = 1$ and $\mu_\alpha = 2$. On the right, we see $V(y(x-y)(x+y))$ and $V(x-y^2)$. Here, $k_\alpha = 2$ and $\mu_\alpha = 3$.

H_g^k is defined analogously. We check if $H_f^k(\alpha) = 0 = H_g^k(\alpha)$. If so, we have proven that $(\alpha, \frac{p(\alpha)}{q(\alpha)})$ is a real intersection point of f and g, thus, the candidate represents this intersection point. If $H_f^k(\alpha) \neq 0$ or $H_g^k(\alpha) \neq 0$, the direct method fails (note that it might still be possible that the curves intersect).

If no failure occurs, we have computed fg-stacks for each critical x-coordinate, and intermediate values in-between, and we know the intersection multiplicity for each intersection in the interior of segments. It should be clear by the explanations that the computed information indeed is correct in all cases. Still, we have to argue why the direct method is successful for curve pairs in generic position.

Lemma 3.3.5. *If the curve pair is in generic position, the direct method does not return a failure.*

Proof. We look at those substeps where a failure can occur, and rule out that this happens for a generic curve pair. First, a failure can happen if a root of r_{fg} is also a root of $\mathrm{lcf}_y(f) \cdot \mathrm{lcf}_y(g) = \mathrm{lcf}_y(fg)$. If fg is generic, its leading coefficient cannot have any real root, so this condition can never be satisfied.

For curve-pair-critical x-coordinates α meaning α is a root of r_{fg} but neither a root of r_f nor r_g, a failure is reported if $k_\alpha = \gcd(f(\alpha,y), g(\alpha,y)) > 1$. Since α is not a root of r_f or r_g, $f(\alpha,y)$ and $g(\alpha,y)$ only have simple roots. If fg is generic, there is only one multiple root of fg at α. That means that exactly one root of $f(\alpha,y)$ and of $g(\alpha,y)$ coincide, and thus, $k_\alpha = 1$.

Another possibility for a failure arises during the elimination of the pairs of a fully-critical x-coordinate α, if the multiplicity of each pair falls below k_α. If fg is generic, there is always one pair that contains the intersection point (α,β) and since $\gcd(f(\alpha,y), g(\alpha,y)) = (y-\beta)^{k_\alpha}$, β is a root of multiplicity of at least k_α for both $f(\alpha,y)$ and $g(\alpha,y)$. It follows that the multiplicity of the isolating interval for β is at least k_α for both $f(\alpha,y)$ and $g(\alpha,y)$. Therefore, the multiplicity of the pair for β is at least k_α.

Finally, the last possibility for a failure is when we detect that $H_f^k(\alpha) \neq 0$ or $H_g^k(\alpha) \neq 0$. If fg is generic, it follows that there exists only one common complex root of f and g at α. Thus, Lemma 2.3.16 is applicable and it follows that $\frac{p(\alpha)}{q(\alpha)}$ is a common root of $f(\alpha,y)$ and $g(\alpha,y)$. Therefore, $H_f^k(\alpha) = 0 = H_g^k(\alpha)$. $\square$

Algorithm 3.8. Curve Pair Analysis, direct approach

INPUT: $f, g \in \mathbb{Z}[x,y]$ square-free and coprime
OUTPUT: Curve pair Analysis for $(V(f), V(g))$, or a flag `REJECT` that denotes that the direct analysis could not be performed

procedure CPA_DIRECT(f,g)
  Call PRE_STACK(f,g) ▷ r_f, r_g, r_{fg} as before
  If for any $\alpha \in V(r_{fg})$, $\mathrm{lcf}_y(f)(\alpha) = 0$ or $\mathrm{lcf}_y(g)(\alpha) = 0$, return `REJECT`
  For each $\alpha \in V(r_{fg})$, compute $k_\alpha := \deg\gcd(f(\alpha,y), g(\alpha,y))$ and $\mu_\alpha := \mathrm{mult}(\alpha, r_{fg})$.
  If for any $\alpha \in V(r_{fg})$ that is not in $V(r_f)$ and not in $V(r_g)$, $k_\alpha \neq 1$, return `REJECT`
  For each $\alpha \in V(r_{fg})$, refine the isolating intervals of the pre-stacks, and assign the minimum of the multiplicities of two paired intervals as the multiplicity of the pair. If the maximal multiplicity over all pairs falls below k, return `REJECT`. Stop as soon as only one pair remains.
  If k_α is odd, or m_α is odd, label the last pair as intersection.
  Otherwise, check whether $H_f^k(\alpha) = 0 = H_g^k(\alpha)$. If so, label the last pair as intersection. Otherwise, return `REJECT`.
  If $k_\alpha = 1$, set the multiplicity of the unique intersection point at α to μ_α
end procedure

We emphasize that if the direct method succeeds, there is no guarantee that the curve pair has been in generic position. Consequently, the preented method does not definitely decide whether a curve pair is generic.

3.3.3. Details of the analysis via a shear

Sheared Analysis The principal idea is as in the one-curve case. We choose a random shear factor s from an integer range where at least half of the possible values lead to a successful analysis. Recall from Corollary 3.2.10 that there are at most $n^4 + n$ bad shear factors for a curve of degree n. As the product fg is of a degree of at most $2n$, there are at most $16n^4 + 2n$ bad shears for the curve pair. For technical reasons that become clear later on, there are additionally $2n^2$ bad directions. Thus, we randomly pick a shear factor from the set $\{1, \ldots, 32n^4 + 4n^2 + 4n\}$, compute the curve analysis of both $V(\mathrm{Sh}_s(f))$ and $V(\mathrm{Sh}_s(g))$, and compute the curve pair analysis of $(V(\mathrm{Sh}_s(f)), V(\mathrm{Sh}_s(g)))$ with the direct approach. If this fails, a new shear factor is chosen.

Backshear process This substep is much simpler than the analogue for the one-curve case. The reason is that intersection points of f and g are invariant under shearing, that is, p is an intersection of $V(f)$ and $V(g)$ if and only if $\mathrm{Sh}(p)$ is an intersection of $V(\mathrm{Sh}(f))$ and $V(\mathrm{Sh}(g))$. Moreover, the intersection multiplicity remains invariant.

For the original curve pair, we have already computed a collection of pre-stacks. In the sheared system, we have full fg-stacks and thus, know all intersections of the sheared curves. We iterate through these intersection points, letting p^* denote a fixed intersection of $V(\mathrm{Sh}(f))$ and $V(\mathrm{Sh}(g))$. We approximate p^* by a box (with rational corners) and explicitly shear back the box into the original system, using the inverse shear transformation $\mathrm{Sh}_{-s}(\,\cdot\,)$. In this way, we can determine the pair that contains p, the preimage of p^* (simply by refining the box until its preimage only overlaps with a single pair – note that a pair can also be interpreted as a box in $\mathbb{R}^2$). This pair is marked as an intersection and its intersection multiplicity is set to the intersection multiplicity of the sheared intersection, if that multiplicity is known. After having done this for each sheared intersection, we know that all remaining unmarked pairs are fake pairs, and they are split by refining the isolating intervals to disjointness.

The method described indeed computes the correct fg-stacks in all cases, but we need an additional postcondition: The intersection multiplicity for intersection points at curve-pair-critical x-coordinates must be determined. It might happen that the algorithm above does not set them properly, even if the curve pair was in generic position. The reason is that intersections at curve-pair-critical x-coordinates might turn into intersections at fully-critical x-coordinates under shearing. If the sheared curve pair is generic, this can only happen if an intersection point becomes critical with respect to a single curve.

Lemma 3.3.6. *If s is chosen such that $(V(\mathrm{Sh}_s(f)), V(\mathrm{Sh}_s(g)))$ is generic, and additionally, no intersection point of regular points of $V(\mathrm{Sh}_s(f))$ and $V(\mathrm{Sh}_s(g))$ has a vertical tangent line, the method just described succeeds in computing the fg-stacks and in determining the intersection multiplicities for intersections in the interior of segments.*

Proof. We have argued that the fg-stacks are computed properly if the curve pair is generic. If the extra condition of the lemma is also satisfied, it follows that no intersection between regular points involves a critical point of a single sheared curve and there cannot be a covertical critical point of a single sheared curve, since the sheared curve pair would not be generic anymore. Hence all such intersections take place at curve-pair-critical x-coordinates and their intersection multiplicities are computed. Since intersections in the interior of segments are, in particular, intersections of regular points, their multiplicity is computed during the backshear step of the algorithm. □

Corollary 3.3.7. *There are at most $16n^4 + 2n^2 + 2n$ shear factors for which the analysis of the curve pair $(V(f), V(g))$ using $V(\mathrm{Sh}_s(f))$ and $V(\mathrm{Sh}_s(g))$ is not successful.*

Proof. There are at most $16n^4 + 2n$ shear factors that bring the curve pair into a non-generic position. Additionally, there are up to n^2 intersection points, and for each of them, one has to rule out that one of the two involved tangent lines becomes vertical. Thus, one has to exclude 2 directions per intersection. □

This causes termination of the algorithm since s is chosen from the range $1, \ldots, 32n^4 + 4n^2 + 4n$ that contains good choices for the shear factor. More precisely, at least every second choice in the range is good and the expected worst-case number of iterations in the sheared method is 2.

3.3.4. Complexity analysis

We turn to the complexity analysis of the presented curve pair analysis algorithm. We will show that for two bivariate polynomials (square-free and coprime), both of magnitude (n, τ), the worst-case bit complexity is

$$\tilde{O}(n^{10}(n + \tau)^2),$$

which equals the complexity of the one-curve analysis.

Algorithm 3.9. Curve pair analysis, shear approach

INPUT: $f, g \in \mathbb{Z}[x, y]$ square-free and coprime, pre-stacks for each critical x-coordinate, shear factor $s \in \mathbb{Z}$

OUTPUT: fg-stacks for critical x-coordinates, or a flag `REJECT` that denotes that the analysis with shear factor s could not be performed

procedure CPA_SHEAR(f, g, s)
  Call PRE_STACK(f, g)
  Call CPA_DIRECT$(\mathrm{Sh}_s(f), \mathrm{Sh}_s(g))$ ▷ This might cause a `REJECT`
  Iterate through all intersection points of the sheared curve pair. Approximate each such point p^* by a box. Shear back the box to determine the unique pair of $(V(f), V(g))$ that contains the preimage p. Mark this pair as an intersection, and assign its multiplicity to that of p^*.
  For all unmarked pairs in the pre-stacks of $(V(f), V(g))$, refine the involved intervals to disjointness.
  If for any $\alpha \in V(r_{fg})$ that is not a root of $V(r_f)$ and not of $V(r_g)$, the multiplicity of any intersection in the fiber is not set, return `REJECT`
end procedure

Algorithm 3.10. Curve Pair Analysis

INPUT: $f, g \in \mathbb{Z}[x, y]$ square-free and coprime

OUTPUT: The curve pair analysis of $(V(f), V(g))$

procedure CURVE_PAIR_ANALYSIS(f, g)
  Call CPA_DIRECT(f, g). On success, return
  $n \leftarrow \max\{\deg f, \deg g\}$
  while true **do**
    Choose $s \in \{1, \ldots, 32n^4 + 4n^2 + 4n\}$ uniformly at random
    Call CPA_SHEAR(f, g, s). On success, return
  end while
end procedure

To derive this complexity bound, we will analyze the algorithm step by step. For many substeps, we can simply refer to analogous steps in the one-curve case for a more compact description.

Complexity of pre-stack computation Finding and merging the critical x-coordinates and computing the intermediate x-coordinates requires $\tilde{O}(n^{10}(n + \tau)^2)$ bit operations, analogous to the projection phase in the curve analysis.

For $\alpha \in \mathbb{R}$, define $\mathrm{sep}_\alpha := \mathrm{sep}((f \cdot g)(\alpha, y))$. If all isolating intervals of fiber points of $f(\alpha, y)$ and $g(\alpha, y)$ are refined to a width of $\frac{\mathrm{sep}_\alpha}{4}$, then no interval of f overlaps with any interval of g, except where the two fiber points form an intersection.

Thus, to construct non-intersection fg-stacks and pre-stacks, we have to isolate the real roots of $f(\alpha, y)$ and of $g(\alpha, y)$ to a width of at most sep_α for any $\alpha \in V(r_f r_g r_{fg})$ and for any $\alpha \in V((r_f r_g r_{fg})')$. By Theorem 2.6.21, this requires $\tilde{O}(n^3(\tau + \log \frac{1}{\mathrm{sep}_\alpha})^2)$ bit operations for each α. Since

$$\sum_{\alpha\in V(r_f r_g r_{fg})} \log\frac{1}{\mathrm{sep}_\alpha} = O(n^3(n+\tau))$$

by Lemma 3.2.5, and

$$\sum_{\alpha\in V((r_f r_g r_{fg})')} \log\frac{1}{\mathrm{sep}_\alpha} = O(n^3(n+\tau))$$

by Lemma 3.2.7, we arrive at a bit complexity of $\tilde{O}(n^9(n+\tau)^2)$ for this step. Thereby, the roots of $r_f r_g r_{fg}$ and its derivative have to be approximated to a precision of at most $O(n^4(n+\tau))$, the costs for that are bounded by $O(n^{10}(n+\tau))$, according to Theorem 2.5.16.

Complexity of the direct analysis In the symbolic precomputation, we have to compute k_α for each critical α which requires evaluating the principal subresultant coefficients of f and g. Analogous to the one-curve case, this can be bounded by $\tilde{O}(n^{10}(n+\tau)^2)$ using Theorem 2.5.24. Computing μ_α is dominated by computing the square-free factorization of r_{fg}, which is not more expensive than computing its square-free part.

The next step is the refinement of intervals that form a pair, until all pairs except one are split, or until the multiplicity of each pair falls below k_α. If there is only one intersection at the fiber, this will be detected at the latest when all intervals are refined to a width of $\frac{\mathrm{sep}_\alpha}{4}$. On the other hand, Lemma 2.6.23 tells us that the multiplicities are set correctly when the precision in the bitstream-Descartes methods is smaller than

$$\mu_0 = \min\left\{\left(\frac{s(f)}{72\cdot 25n^2}\right)^n, 2^{-7n}\right\},$$

thus, if there is more than one intersection, this will eventually be detected for such a precision. In any case, after

$$\tilde{O}(n^3(\tau + \log\frac{1}{\mathrm{sep}(f(\alpha,y))} + \log\frac{1}{\mathrm{sep}(g(\alpha,y))} + \log\frac{1}{\mathrm{sep}_\alpha})^2)$$

bit operations, a decision will have been made. Since the sum of the separations is again bounded by $O(n^3(n+\tau))$ for all three summands, it follows that after a total running time of $\tilde{O}(n^9(n+\tau)^2)$, either each pre-stack has been reduced to one pair or a failure has been reported.

The last step is to verify the existence of a candidate, in case the previous step was successful. For that step, we can use the same complexity analysis as in Section 3.2.4 for the singular point detection: the polynomial H_f^k is of magnitude $(n^3, n^2(n+\tau))$. Testing $H_f^k(\alpha) = 0$ for all $\alpha \in V(r_{fg})$ requires $\tilde{O}(n^{10}(n+\tau))$ operations, and doing so for each $k = 0, \ldots, n$ gives $\tilde{O}(n^{11}(n+\tau)) = \tilde{O}(n^{10}(n+\tau)^2)$.

Complexity of the analysis with a shear Shearing the curve pair changes the bitsizes of the polynomial coefficients only by a polynomial factor in s. Because $s = O(\log n)$, the analysis of a sheared curve pair is not more expensive than for the unsheared curve pair (in $\tilde{O}$-notation). The expected number of such sheared pair analyses until we arrive at a generic position for which the analysis works is two.

It remains to bound the backshear process. This is analogous to the one-curve analysis. Each sheared intersection point has to be refined to $O(n^3(n+\tau))$ bits precision, until the x-coordinate of its preimage can be determined. If its y-coordinate is refined to a width of $\frac{\mathrm{sep}_\alpha}{4}$, where α is the x-coordinate of the preimage, then the pair which the intersection belongs to is uniquely determined. But $\mathrm{sep}_\alpha = O(n^3(n+\tau))$ (since the sum over all sep_α's is bounded by this quantity), so it suffices to refine the x- and y-coordinates of each sheared intersection point to $O(n^3(n+\tau))$ bits. This requires $\tilde{O}(n^{10}(n+\tau)^2)$ bit operations, as in the case of sheared event points in the one-curve case.

Splitting the fake pairs in the final step again requires refining each isolating interval to a width of $\frac{\mathrm{sep}_\alpha}{4}$ in the worst case; the complexity for this has already been analyzed.

After all, we conclude that all substeps of the curve analysis algorithm are dominated by

$$\tilde{O}(n^{10}(n+\tau)^2)$$

as claimed at the beginning of this section.

3.3.5. The complexity of the sweep-line algorithm

With the complexity bounds for curve analysis and curve pair analysis, we derive a worst-case bit-complexity bound for the sweep-line algorithm applied to algebraic curves. We assume that our input objects are polynomials $f_1, \ldots, f_m$, all of magnitude (n, τ), and pairwise coprime. We will prove that the dominant operation is the computation of the $\frac{m(m-1)}{2}$ curve pair analyses.

Theorem 3.3.8. *The arrangement induced by $V(f_1), \ldots, V(f_m)$ can be computed with*

$$\tilde{O}(m^2 n^{10}(n+\tau)^2)$$

bit operations.

Proof. We assume that a curve analysis for each $V(f_i)$, and a curve pair analysis for each pair $(V(f_i), V(f_j))$ is available. Computing them can be done within the required complexity bound, according to the complexity results of this chapter. The remainder of this proof shows that the sweep-line method itself does not increase the complexity.

The sweep-line algorithm for linear segments has a complexity of $O((s+I)(\log s))$, where s is the number of segments and I is the number of intersection points ([dBvKOS00, §2.2], [MN00, §10.7]). The geometric primitives in the algorithm are assumed to have constant running time for linear segments. Thus, introducing an upper bound P for the cost of one evaluation of a geometric primitive, we immediately obtain $O(P(s+I)(\log s))$ as an upper bound for the sweep-line algorithm in the curved case.[25]

We have to give upper bounds for s, I, and P. By Corollary 2.2.15, the maximal number of segments of a curve is n^3, so $s \leq mn^3$. Since two curves intersect in up to n^2 many points, it holds that $I = O(n^2 m^2)$. For P, reconsider the realization of the geometric primitives at the end of Section 3.1.4. The most expensive one is the intersections primitive, where we have to go through the $O(n^2)$ intersection points of the curve pair, and compare the points

[25] When switching from the linear to the curved case, the reordering of segments at intersection points becomes more complicated. However, thanks to our knowledge of intersection multiplicities, we can use [BK07] to get a running time proportional to the number of involved segments, just as in the linear case.

with the involved segments by calling **Compare_y_at_x**. It is not hard to see that each such call is linear in the number of points in the fg-stack at the intersection point, if the curve pair analysis is known. Accordingly, as a rough estimate, we can bound $P = O(n^3)$. Therefore, we get a total complexity of $O(n^3(n^3m + n^2m^2)\log nm) = \tilde{O}(mn^5(n+m))$ for sweep-line algorithm itself, which is clearly dominated by the cost of the curve pair analyses. □

The complexity bound for linear segments depends adaptively on the number of actual intersection points, whereas the bound for algebraic segments just derived does not. We lose this property because we assume that each curve pair must be analyzed during the algorithm. When considering complete curves, this assumption is not unrealistic; for instance, if all curves have odd degree, each pair of curves has a real intersection point (except for degenerate examples), and a curve pair analysis is necessary to compute this intersection point. Still, if we restrict the arrangement to a subset of the input segments, this may not longer be the case.

Corollary 3.3.9. *Let $f_1, \ldots, f_m$ be as above. Consider a subset S of the up to mn^3 x-monotone segments induced by the curves $V(f_1), \ldots, V(f_m)$. Let C be the number of curve pair analyses that are triggered to compute the arrangement. The complexity is then bounded by*

$$\tilde{O}((m+C)n^{10}(n+\tau)^2).$$

Proof. We still have to perform m curve analyses (for each curve), and C curve pair analyses, which exactly matches the complexity bound.

The sweep-line algorithm has the same complexity as before, namely $O(P(s+I)\log s)$. Now, $s \in O(n^3m)$, $I \in O(n^2C)$, and $P \in O(n^3)$. This yields $O(n^3(n^3m + n^2C)\log nm) = \tilde{O}(n^6(m+C))$, which is again clearly dominated by the complexity bound. □

This bound depends on the rather abstract quantity C. When is a curve pair analysis of $(V(f), V(g))$ "triggered" by the algorithm? Of course, this happens when a primitive that involves $V(f)$ and $V(g)$ is called. This happens as soon as a segment of $V(f)$ and a segment of $V(g)$ become neighbors on the status line (the sweep-line). For practical purposes, it is desirable to decrease the number of triggered curve pair analyses as much as possible. A simple method to avoid curve pair analyses in practice is discussed in Section 4.2.3.

Summary

The curve pair analysis is the main tool to compute the vertical ordering of two curves at any fiber and, in particular, to compute their intersections. An algorithm that shows the same complexity bound as the curve analysis has been presented. With that, we can derive the first known complexity bound for exact arrangements of arbitrary algebraic plane curves.

Programming without an overall architecture or design in mind is like exploring a cave with only a flashlight: You don't know where you've been, you don't know where you're going, and you don't know quite where you are.

Danny Thorpe

4 Implementation of Arrangements of Algebraic Plane Curves

The algorithms from the previous chapter solve the problem of computing the arrangement of algebraic curves, with a complexity bound that is currently the best available. However, an "efficient" algorithm should also show a satisfactory practical performance. In order to verify the practical efficiency of our approach, we have implemented both the curve analysis and the curve pair analysis, as described in Chapter 3. Nevertheless, we slightly deviate from the description in several places, and this is described in Section 4.1. One reason is that tuning sub-operations in the algorithm may drastically improve the practical running time of the algorithm but make the analysis much harder without improving the overall bound. As an example of that, we mention the usage of modular methods in our algorithm. A second reason is that explaining the algorithm in full detail from scratch makes it very difficult to follow the overall structure. Hence, we aimed for a "minimal" version in Chapter 3 that still captures the esence of our implementation, and postponed the optimized method for several special situations to this chapter.

The implementation of curve and curve pair analysis (with variations) is embedded into the context of the CGAL library. In fact, the layered approach for arrangement computation (reduce the sweep-line algorithm to the implementation of geometric primitives; reduce the geometric predicates to the implementation of curve analysis and curve pair analysis) can be modelled into software by defining proper *concepts* and *models*. We will briefly introduce CGALand describe the software framework used in Section 4.2. Finally, we measure the practical performance of the arrangement algorithm via extensive experiments in Section 4.3

4.1. Implementation of curve and curve pair analysis

We discuss several differences between the theoretical formulation and its practical implementation. (Almost) all these changes are guided by the aim of practical optimization. They are mainly independent of each other; some optimizations can be switched on or off in our implementation using appropriate compiler flags.

4.1.1. Intermediate stacks revisited

Certainly one of the most striking differences between the theoretical formulation and its practical implementation is how intermediate stacks are obtained. Recall that we stated in Section 3.2.2 that when choosing rational intermediate values between critical x-coordinates of a curve, we were only able to show a complexity bound of $\tilde{O}(n^{12}(n+\tau)^2)$, which is much worse than the rest of the curve analysis algorithm. Therefore, we switched to the strategy of selecting roots of the derivative of the resultant polynomial instead, for which we were able to use the same techniques as for the critical x-coordinates in the analysis.

In the implementation, we stick to the theoretically inferior method of choosing rational x-coordinates (of possibly small bitsize) as intermediate values. In fact, our experiments have shown that the computation time of the intermediate stacks is never a dominant factor off the algorithm (for the curve analysis, it takes <1% in all cases). We believe that the inferior worst-case behavior is an artefact of the analysis and that especially the separation at such an intermediate fiber is highly overestimated, although we failed to show an improved bound.

Because introducing the derivative of the resultant in the algorithm in practice causes more non-trivial computations (isolating the roots, merging them with the resultant roots), we claim that using rational intermediates indeed improves the practical performance, without having actually implemented the alternative.

4.1.2. The implementation of the bitstream-Descartes method

We already mentioned in Section 2.6 that there are two versions of the bitstream-Descartes method known in the literature. For theoretical considerations, we have chosen to discuss the approach by Mehlhorn and Sagraloff [MS09] because the results about runtime complexity follow in a more direct way. However, no C++ implementation of their algorithm is currently available. Instead, we use the approach of Eigenwillig [Eig08] for our implementation. Eigenwillig provides a `bitstream_tree` class, which mainly represents the subdivision tree for a bitstream polynomial f and assigns a set of possible sign variations to each interval in the subdivision (recall that Eigenwillig's approach considers all approximations of f at once; therefore, it returns a set of sign variations). This tree can be explored to any depth; the precision of the bitstream coefficients is controlled internally and is adaptively increased when the tree is explored deeper; see [Eig08] for details. Using this trees allows the convenient realization of all variants of the bitstream-Descartes method and further refinements of isolating intervals, introduced in Section 2.6.

4.1.3. Faster methods for simple resultants roots

Our algorithms for the curve analysis and curve pair analysis are designed with the goal in mind of reducing symbolic computations to a minimum, and replacing them by fast but certified methods. We can further decrease the amount of symbolic operations by introducing a special treatment for simple resultant roots, that is, roots of multiplicity one. This holds true for both the curve and curve pair analyses, although the ways of exploiting the simpleness of a resultant root are quite different. Therefore, we treat both cases separately. The situation of simple resultant roots appears quite frequently, also for highly degenerate curves and curve pairs. Therefore, the presented optimizations have a high impact in practice.

Simple resultant roots in curve analysis: We use the following lemma from Wolpert [Wol, §4.1.1].

Lemma 4.1.1. *Let (α, β) be an intersection point of $V(f)$ and $V(g)$. Then α is a multiple root of $\mathrm{res}_y(f, g)$ if and only if*

$$(\frac{\partial f}{\partial x} \cdot \frac{\partial g}{\partial y} - \frac{\partial f}{\partial y} \cdot \frac{\partial g}{\partial x})(\alpha, \beta) \cdot \mathrm{sres}_1(f, g)(\alpha) = 0.$$

Consider a simple root α of $\mathrm{res}_y(f, \frac{\partial f}{\partial y})$ that is not a root of $\mathrm{lcf}_y(f)$. First of all, there is exactly one critical point in the fiber over α. From Lemma 4.1.1, it also follows that $\mathrm{sres}_1(f, \frac{\partial f}{\partial y})(\alpha) \neq 0$, thus, $k_\alpha := \deg \gcd(f, \frac{\partial f}{\partial y}) = 1$ by Lemma 2.3.14. Moreover, Lemma 4.1.1 reveals that

$$(\frac{\partial f}{\partial x} \cdot \frac{\partial f}{\partial yy})(\alpha, \beta) \neq 0,$$

and so, (α, β) is not singular (since $\frac{\partial f}{\partial x}$ does not vanish), and x-extreme (since $\frac{\partial f}{\partial yy}(\alpha, \beta)$ does not vanish). This implies that the stack at α consists of exactly one non-singular x-extreme point and a certain number of regular points. Thus, the number of fiber points in the left neighboring and right neighboring intermediate stacks differ by exactly two; let these numbers be denoted by $m - 1$ and $m + 1$, respectively. Then, the fiber over α must consist of precisely m points.

To summarize, for simple resultant roots, we can deduce $k_\alpha = 1$ directly, and m_α, the number of fiber points, is easily computable by looking at the number of fiber points in the adjacent intermediate stacks. This means that we do not have to evaluate the principal Sturm-Habicht coefficients at α, as it is formulated in Step 6 of Algorithm 3.3, in order to get the input for the m-k-bitstream-Descartes algorithm. This further reduces the amount of symbolic computations in this particular situation.

Simple resultant roots in curve pair analysis: Let μ_α be the multiplicity of α in $\mathrm{res}(f, g)$. We assume that $\mu_\alpha \in \{0, 1\}$. The case $\mu_\alpha = 0$ is of interest for curve pairs, because we also need the fg-stacks at one-curve-critical x-coordinates. We assume that all intermediate stacks of the curve pair $(V(f), V(g))$ are already available. We first discuss two examples to demonstrate the principal idea, and then give the complete description at the end of the paragraph.

Assume first that $\mu_\alpha = 1$ and that α is neither critical for $V(f)$ nor $V(g)$. This means that all fiber points have branch numbers $(1, 1)$. Also, there is precisely one intersection point and as it is a simple one; the involved segments have to change sides. In this situation, we can produce the fg-stack without producing the f-stack or the g-stack at α (as it was proposed in Algorithm 3.7). Instead, we are comparing the arc pattern at the left and right neighboring intermediate fg-stacks, as exemplified in Figure 4.1.

Assume now that $\mu_\alpha = 0$. Recall that Algorithm 3.7 proposes obtaining the fg-stack by getting the f-stack and the g-stack at α and refining the isolating intervals to disjointness (Step 5). This is certainly a correct and fairly simple way of obtaining the stack, but it

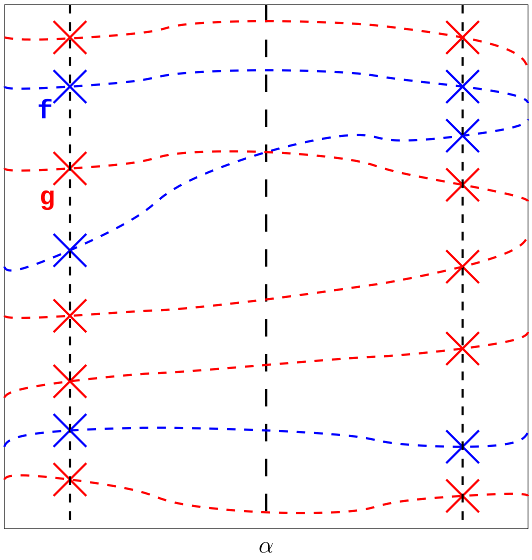


Figure 4.1. The left intermediate fg-stack is "gfggfgfg", and the right intermediate stack is "gfggffgg". The fiber points must be connected according to the dashed lines, thus, one can observe that the fg-stack at α is "gfggfig".

has a hidden drawback. For instance, if α is a critical x-coordinate for $V(f)$, but not for $V(g)$, the g-stack at α does not exist yet and must first be produced using the bitstream-Descartes method. Also, if $f(\alpha, y)$ and $g(\alpha, y)$ have a bad separation, a lot of refinement is necessary in practice in order to separate the isolating intervals. Thus, the goal is to avoid the creation of the g-stack entirely, if possible, and to avoid as much as possible refinement until disjointness.

The idea is to combine the knowledge from the left and right neighboring intermediate stacks with the branch numbers of each fiber point at α to deduce the fg-stack at α without further refining. In Figure 4.2, we provide an example where this is possible.

The two examples given above were rather well-behaved. If $\mu_\alpha = 1$, the situation can also be more complicated; for instance, the curves might intersect at an x-extremal (but non-singular) point of $V(f)$. Also, there might be an arbitrary number of covertical singularities of either curve or vertically asymptotic arcs towards α. Also for $\mu_\alpha = 0$, the example from above is not general enough. For instance, if $V(f)$ has an isolated point at α, we cannot deduce the fg-stack at α without computing the fiber of $V(g)$ at α.

The general algorithm works both for $\mu_\alpha = 0$ and $\mu_\alpha = 1$: For $V(f)$, we know the number of points in the fiber at α (either because an f-stack is available or because it is the same as for the neighboring intermediate stacks); same for $V(g)$. Let $f_1, \ldots, f_{m_1}$ and $g_1, \ldots, g_{m_2}$ denote these fiber points at α, sorted from the bottom upwards. We want to merge these two sequences, which means that we iterate through both lists and find the minimum of f_i and g_j in each step until the end of both lists is reached. The fg-stack at α is initialized with an empty string and is built up during the merge.

For each f_i and each g_j, we know which arcs at the left and right neighboring intermediate stacks are connected with it (again, either by the f-stack at α or because the branch numbers are all $(1, 1)$ if no f-stack is available). For a concrete f_i and g_j, we call them

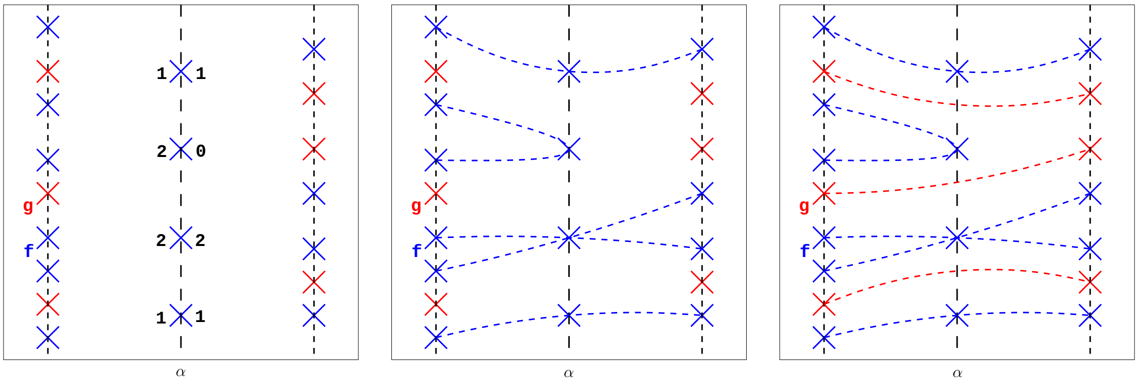


Figure 4.2. Assume α is a critical x-coordinate of $V(f)$, not of $V(g)$. We know the fg-stacks at the left and right and the branch numbers of $V(f)$ at α (left picture). The center picture translates this combinatorial information into a geometric picture. The red points must be connected by x-monotone segments which are neither intersecting each other nor intersecting $V(f)$. Thus, only connections as depicted on the right are possible and thus, the fg-stack is "fgfgfgf".

involved arcs, and their bottom-upward ordering induces an *involved arc pattern* on the left and on the right. Both arc patterns involved are substrings of the fg-stacks at the corresponding intermediate positions.

There are three possible cases. First, the involved arcs might show one of the *intersection patterns*, which is "f" on the left and "gfg" on the right, or "fg" on the left and "gf" on the right (or any analogous pattern with "g" and "f", or "left" and "right" interchanged – see also the left of Figure 4.3). In such cases, we have found an intersection of $V(f)$ and $V(g)$. We add "i" to the fg-stack at α and pass to the next f_i and g_j.

If no intersection pattern has been recognized, $f_i \neq g_j$ (no intersection). Thus, we have to determine which point comes first in the vertical ordering. If the involved arc pattern on the left (or on the right) contains both an "f" and a "g", the first entry in the involved arc pattern determines the minimum. Thus, if the corresponding arc pattern starts with "f", we add "f" to the fg-stack and pass to the next f_i (leaving g_j unchanged for the next iteration). If it starts with "g", we add "g" to the fg-stack and pass to the next g_j.

It remains the case that the involved arc patterns do not determine the minimal point (this happens, for instance, if f_i is an isolating point, or if f_i and g_j are both x-extreme but their segments lie on different sides – see the right of Figure 4.3). In this case, we construct the f-stack and the g-stack at α, if they do not exist yet, and refine the isolating intervals of f_i and g_j to disjointness.

To summarize, we have derived a method to compute fg-stacks at critical x-coordinates α with $\mu_\alpha \leq 1$. The method is guaranteed to work for any α (no failure) no matter what the level of degeneracy of a single curve at α. In contrast to the general method, we avoid to evaluate the principal subresultant coefficients at α. In many cases, we do not even have to construct an f-stack at the critical position (what we always need are f-stack, g-stack, and fg-stack at intermediate positions between critical x-coordinates, but by Section 4.1.1 this is expected to be cheap in practice).

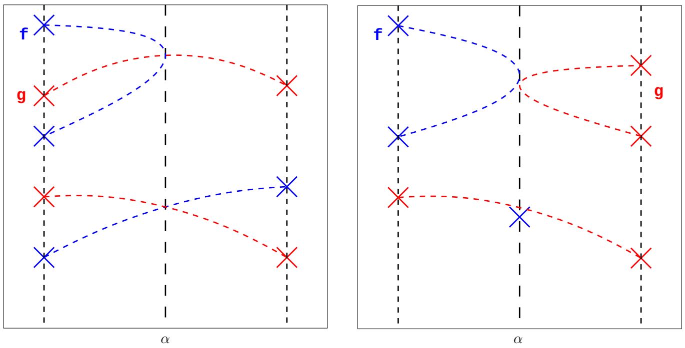


Figure 4.3. On the left: the two intersection patterns. On the right: two examples of where the f-stack and the g-stack are necessary.

These ideas can also be used as an additional filter in another step of the curve pair analysis algorithm. For $\mu_\alpha > 1$, recall the general strategy to construct an fg-stack. We compute a pre-stack, identify a candidate intersection point, and verify that the candidate is an intersection. The verification step first checks whether k_α or μ_α is odd (Lemma 3.3.4), and if not, tests for an intersection symbolically. We introduce another filter between these two steps: if k_α and μ_α are both even, we look at the arc pattern involved, both on the left and on the right. If any of these contains a substring of the form "fg^{i}f" or "gf^{i}g" with $i \geq 1$ (where "f^i" denotes the concatenation of i "f"'s), the candidate is an intersection point. The reason is that one segment of the one curve, say $V(f)$, is sandwiched between two segments of $V(g)$, so they have to intersect at α.

4.1.4. Avoiding a shear for a non-generic curve

Recall the high-level strategy of our curve analysis: Try a direct analysis first, perform an analysis of a sheared curve in case of a failure, and finally shear back into the original system. We have shown that the complexity does not increase by analyzing a sheared curve instead. However, shearing has a bad effect on the practical performance, since the coefficient size grows considerably (this is hidden in $\tilde{O}$-notation). Also, a shear destroys the sparseness of polynomials and results in more symbolic computations in the algorithm (besides the resultant in the original system, two resultants in the sheared system are needed also). After all, shearing should be avoided as much as possible. Already our algorithm is guided by this thought, since it first tries a direct approach in the original system (and does not initially shear to bring the curve into a possibly generic position).

In Algorithm 3.3, we pass to a sheared curve as soon as we detect a root of the leading coefficient of $f \in \mathbb{Z}[x][y]$, due to potential vertically asymptotic arcs. We show that sometimes such a shear can be avoided, based on the following observation.

Lemma 4.1.2. *Let $f \in \mathbb{Z}[x][y]$ with $\deg_y(f) = n$ and coefficients $a_n, \ldots, a_0 \in \mathbb{Z}[x]$. Let α be a simple root of $\mathrm{lcf}_y(f) = a_n(x)$ and also a simple root of $R := \mathrm{res}_y(f, \frac{\partial f}{\partial y})$. Then, the fiber of $V(f)$ at α consists only of non-critical points, in other words, $f(\alpha, y)$ is square-free. Moreover, there is exactly one vertically asymptotic segment s_ℓ on the left and exactly one vertically asymptotic segment s_r on the right. If $a_n'(\alpha) \cdot a_{n-1}(\alpha) < 0$, s_ℓ diverges to*

$(\alpha, -\infty)$ and s_r diverges to $(\alpha, +\infty)$. Otherwise, if $a_n'(\alpha) \cdot a_{n-1}(\alpha) > 0$, s_ℓ diverges to $(\alpha, +\infty)$ and s_r diverges to $(\alpha, -\infty)$.

Proof. For the fact that $f(\alpha, y)$ is square-free, we give an intuitive argument. Assume that there is a critical point p at α. Consider a small perturbation of the coefficients of f that leaves the leading coefficient untouched. The perturbed curve has a (complex) critical point close to p, at an x-coordinate $\alpha' \in \mathbb{C}$. Thus, the perturbed curve has critical x-coordinates at α and α'. By continuity, R must have at least a double root at α, which contradicts the assumptions.

We next argue that there is exactly one vertically asymptotic arc each of the left and right. Consider $f^* := y^n f(x, \frac{1}{y})$. Then $(\alpha, 0)$ is a point on $V(f^*)$, and any segment that enters $(\alpha, 0)$ corresponds to a vertically asymptotic segment of $V(f)$ (Figure 4.4). It is enough to show that $(\alpha, 0)$ is a non-critical point of f^*.

Note that $a_{n-1}(\alpha) \neq 0$, because otherwise, it can be easily seen that α is a multiple root of R (the first two columns of the Sylvester matrix would vanish). Therefore,

$$\frac{\partial f^*}{\partial y}(\alpha, 0) = a_{n-1}(\alpha) \neq 0.$$

Moreover,

$$\frac{\partial f^*}{\partial x}(\alpha, 0) = a_n'(\alpha) \neq 0,$$

because α is a simple root of a_n. Therefore, $(\alpha, 0)$ is indeed non-critical. Moreover, since both partial derivatives do not vanish, $V(f^*)$ crosses the x-axis at $(\alpha, 0)$. This implies that one of the asymptotic arcs goes to $-\infty$ and the other goes to $+\infty$ (just consider the back transformation which maps (x, y) to $(x, \frac{1}{y})$). The curve changes from $y < 0$ to $y > 0$ if and only if the gradient vector $(a_n'(\alpha), a_{n-1}(\alpha))$ lies in the second or fourth quadrant of the coordinate system, which is exactly the case when $a_n'(\alpha) \cdot a_{n-1}(\alpha) < 0$. $\square$

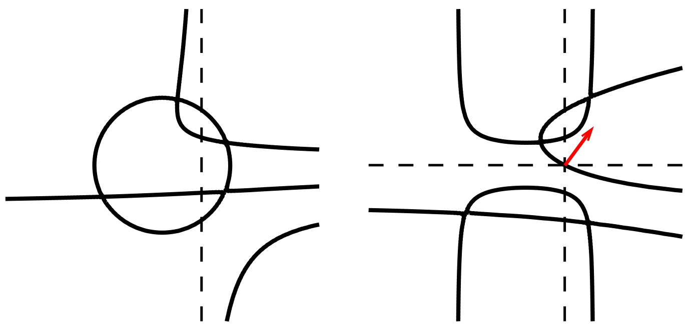


Figure 4.4. On the left: The curve $V(((x-1)y^3 + (x+1)y^2 - 1)(x^2 + y^2 - 3))$,which has a vertical asymptote at $x = 1$ (dashed). On the right: The curve $V(f^*)$. Note that the point $(1, 0)$ is an element of $V(f^*)$, and the gradient points into the first quadrant. Indeed, the gradient of $(1, 0)$ in f^* is $(1, 1)$.

How can we use this in the curve analysis algorithm? Whenever we encounter an α that satisfies the prerequisites of Lemma 4.1.2, we apply the (square-free) bitstream-Descartes method on α to construct the fiber (and all branch numbers are set to $(1, 1)$).

The number of vertical asymptotes (compare Definition 3.1.6) are set either to $(1, 0, 0, 1)$ if $a_n'(\alpha) \cdot a_{n-1}(\alpha) < 0$, or to $(0, 1, 1, 0)$ otherwise.

We sketch a further approach to avoid shearing.[26] The idea is to compute a possible set of isolating intervals for the fiber at α and to verify these intervals by finding a segment of the curve that enters each interval. The method that we describe only works in the absence of isolated vertices – it does not terminate otherwise. Also, the method is not implemented; we mention it anyway because it might be extendible to an efficient method that works without shearing.

Let α be a critical x-coordinate of $V(f)$. As before, we compute m, the number of fiber points over α, using the principal Sturm-Habicht coefficients. Next, we start the bitstream-Descartes subdivision – we do not give full details here, but similarly to the oracle-bitstream-Descartes method, we identify clusters that constitute candidates for isolating intervals. Once we find m disjoint clusters $I_1, \ldots, I_m$, we take them as candidates for isolating intervals. We choose rational intermediate values $q_0, \ldots, q_m$ that separate the clusters from each other, and from $-\infty$ and $+\infty$, which means that $q_0 < I_1 < q_1 < \ldots < I_m < q_m$. Next, we pick an x-interval $[c, d]$ containing α such that $V(f)$ does not intersect any horizontal line $y = q_i$ over any point in $[c, d]$. This interval is found as follows: Let $[c, d]$ be the isolating interval of α. Evaluate $\square f([c, d], q_i)$ using interval arithmetic, for any $i = 0, \ldots, m$. If any of the intervals contains zero, refine the interval. Otherwise, $[c, d]$ has the desired property.

We compute the fiber of $V(f)$ at c and d. If any fiber point at c or d lies between q_{i-1} and q_i, it is certified that the interval I_i contains a root (because the segment must cross the fiber $x = \alpha$, but it cannot cross the lines $y = q_{i-1}$ and $q = q_i$). If all intervals can be certified in this way, they are indeed isolating intervals. Otherwise, if some interval has not been certified, we throw it away and subdivide the remaining intervals until m disjoint intervals show up again (in this last step, we require that no isolated vertices are present, since we might potentially remove an isolating interval). See also Figure 4.5 for an example.

We remark that this method already provides enough information to compute the whole f-stack, in the case of success. The branch numbers are determined by the number of fiber points at c and d within the corresponding region $[q_{i-1}, q_i]$, and even the number of vertically asymptotic arcs is immediately apparent, by the number of fiber points in the regions $(-\infty, q_0)$ and $(q_m, +\infty)$.

Finally, we wish to mention that we are aware of a method of extending this approach to curves with isolated vertices. The idea is mainly to isolate the roots in the complex fiber at α. We skip further descriptions because this is work in progress, and we are far from an efficient implementation yet.

4.1.5. The choice of the shear factors

In the preceding section, we considered how to avoid shearing. Still, degenerate situations require a change of coordinates. In such cases, we try to reduce the coefficient swell-up and the number of anaylses of sheared curves.

The first question in this context is how to choose the shear factors in practice. Algorithms 3.6 and 3.10 propose picking a random integer from a range $\{1, \ldots, \max_s\}$, where

[26] Michael Sagraloff, personal communication.

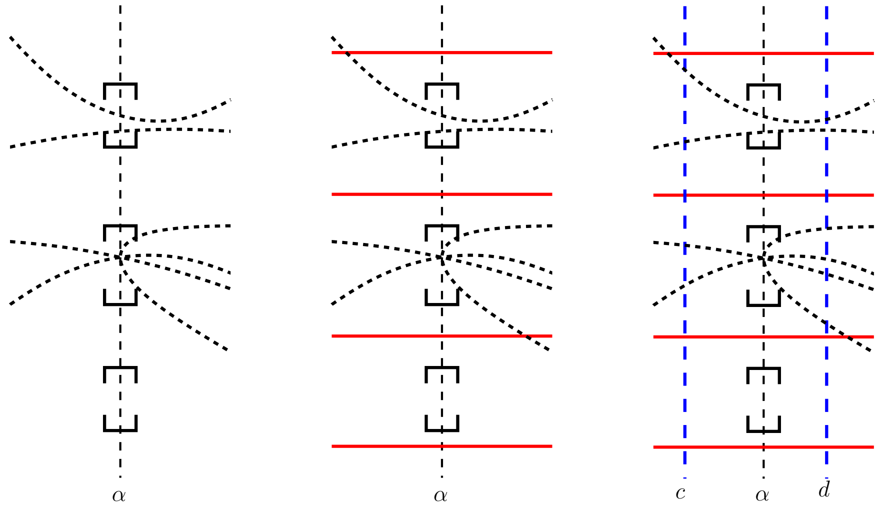


Figure 4.5. Assume that $m = 3$ and that the three intervals on the left picture are computed as clusters, at some stage in the subdivision. We choose rational intermediate values (middle picture) and compute c and d such that the curve (dashed) does not intersect any intermediate line in $[c, d]$. We see that the upper two intervals are certified to contain a root, but the lower one is not. Thus, the algorithm would further subdivide the upper two intervals until one of them is split into two.

$\max_s = O(n^4)$. This bound was obtained by counting the number of possible "bad" shear factors for a given curve or curve pair. However, it is very unlikely that all such bad shear factors are integer values. In practice, a small integer will put the curve (pair) into a generic position, except for ill-conditioned examples. A small shear factor has computational advantages since the swell-up of the coefficients is reduced.

We have implemented the following strategy to pick random shear factors. We set $\max_s := 8$, and start with the range $\{1, \ldots, \max_s\}$. We pick shear factors at random, counting the number of shear factors used so far. Whenever this number exceeds $\max_s/2$, we double $\max_s$. Although this does not theoretically guarantee a constant expected number of shear transformations needed during the analysis, it is guaranteed that eventually a good s will be found, and small shear factors are more likely to achieve this than big ones by this strategy.

We turn to a related problem. Assume that an arrangement of m curves is computed, and each curve passes through the two common points (α, β_1) and (α, β_2). This means that each curve pair is analyzed using a sheared transformation of both curves. If a random shear factor is chosen separately for each curve pair, this might lead to more analyses of sheared curves than actually necessary (if $m \gg 8$, it is likely that each curve is sheared by any shear factor $s = 1, \ldots, 8$ during the algorithm).

To overcome such problems, all curves and curve pairs initially agree on a unique (infinite) list of shear factors $(s_1, s_2, \ldots)$ that is built according to the strategy above. When a curve or a curve pair is not in generic position, the curves are sheared by shear factor s_1, then by s_2, and so on until the analysis is successful. In that way, a few transformations per curve will usually suffice.

4.1.6. Modular methods

A last, but certainly not least, optimization involves the use of modular methods in the algorithm. Most importantly, we use a modular algorithm to compute the greatest common divisor of integer polynomials, described in [HH09], instead of Algorithm 2.6. Implementing the gcd with a modular algorithm has a tremendous effect on practical efficiency, especially for higher degrees. This has already been demonstrated in [Ker06], where the gcd algorithm from the NTL[27] has been used.

Another optimization is entailed by *modular filters*. Remember that the curve analysis expects a square-free polynomial as input and the curve pair analysis requires coprime polynomials. In most cases, these requirements are satisfied and we aim for a quick verification. For that, the gcd of the two polynomials (or in the one-curve case, the gcd of the polynomial and its derivative) is computed in the domain $\mathbb{Z}_p[x, y]$, where p is some prime number. If their gcd is a constant in $\mathbb{Z}_p[x, y]$, it is certified that it is also a constant in $\mathbb{Z}[x, y]$. If, however, the degree is larger, no definite statement can be made, and the algorithm passes to exact computation over $\mathbb{Z}[x, y]$.

A further use of modular methods appears when symbolically checking for an intersection point in the curve pair analysis (page 138), as well as for the optional computation of the singular flag in Section 3.2.4. In both cases, we compute a polynomial

$$H_f^{(k)}(x) := q(x)^{\deg_y(f)} f(x, \frac{p(x)}{q(x)}) \in \mathbb{Z}[x]$$

and check whether it vanishes at some α. The computation of the polynomial $H_f^{(k)}$ is expensive – however, because we only checked whether it vanishes at α afterwards, we do not have to compute it completely. With R being the defining polynomial for α, we can instead compute $H_f^{(k)}$ in the ring $\mathbb{Z}[x]/R$ which limits the degree of H_f^k (but increases the coefficient sizes). We observed a significant speed-up when working in this modular ring (which is not necessarily a domain because R is not assumed to be irreducible).

Summary

The algorithms for curve analysis (Algorithm 3.6) and curve pair analysis (Algorithm 3.10) can be tuned at numerous places in order to speed up their practical performance. In particular, we have presented methods to handle simple resultant roots more efficiently and methods to avoid curves having to be sheared in some situations. Also, using a common order of shear factors for all curves, and the use of modular arithmetic results in significant speed-ups.

4.2. Software design in CGAL

CGAL [CGA08], the Computational Geometry Algorithms Library, is the state of the art in implementing geometric algorithms. While in the past the major focus of the library was on linear geometry, its developers decided to direct more of the project's attention to non-linear geometry [EKP+04] [PT08], and our work can be seen as a contribution to this.

[27] http://www.shoup.net/ntl/

The CGAL library follows the *generic programming paradigm*. This allows the exchange of types, that is, an algorithm (or data structure) can be parameterized by any type as long as this type fulfills a certain set of requirements. Such a set of requirements is called a *concept*. A data type that meets the requirements of a concept is called a *model* of this concept. For readers unfamiliar with generic programming, the following analogue from mathematics should be useful: a group is a concept and a ring is a refined concept of group. A specific ring, for instance, $\mathbb{Z}$, is a model of the concept ring. In C++ code, types and algorithms can be written in a generic way by class templates whose template arguments are certain concepts, and models for these concepts are instantiations of these template parameters.

Algorithms in CGAL are usually parameterized by a *traits class*, also called a *kernel*.[28] A traits class encapsulates the basic geometric types (and the operations on them) that are expected by a combinatorial algorithm. A good example for that is a geometric algorithm, whose traits class defines the input objects the algorithm is running on and the geometric primitives that are needed to process the objects. Of course, a traits class can itself depend on other primitives, which are composed in another traits class.

The kernels that we provide all follow the *exact geometric computation paradigm* (EGC paradigm), presented in the introduction of this thesis (page 8): Each geometric predicate must compute the mathematically correct result in all cases. Hence, a (complete) combinatorial algorithm based on these predicates always computes the correct solution. Clearly, we need adequate number types, modelling integers and rationals of arbitrary size to ensure exactness for all input curves.

4.2.1. Arrangements of algebraic curves in CGAL

We make the rather abstract terms like kernel, concept and model concrete by explaining the relevant software classes to realize arrangements induced by arbitrary algebraic plane curves. Most of the new classes presented have a prototypical status and their integration into the CGAL library is ongoing work; for instance, the latest CGAL release, 3.4, contains support for polynomials [Hem08], which is essential for our contribution. Release 3.5, which is currently in preparation, will contain the concepts and models for `Algebraic_kernel_1` and `Algebraic_kernel_2` as described below.

Omitting some template parameters that are irrelevant for our considerations, the arrangement computation for algebraic curves is realized by the following hierarchy of template classes:[29]

- `template<class ArrangementTraits_2> class Arrangement_2;`
- `template<class CurveAnalysisTraits_2> class Curved_kernel_via_analysis_2;`
- `template<class AlgebraicKernel_1> class Algebraic_kernel_2;`
- `template<class Coefficient> class Algebraic_kernel_1;`

In this list, an instantiation of each class is a valid template argument of the class in the preceding line. For instance, `Curved_kernel_via_analysis_2` is a model of the concept `ArrangementTraits_2`. We sketch the content and the function of each class next. We will see that the presented hierarchy precisely corresponds to the different layers in the arrangement algorithm as described in Chapter 3.

[28] These two terms have slightly different meanings. A `traits class` is usually stateless, whereas a *kernel* is allowed to store member variables. This distinction is not important for our considerations.

[29] Note the naming convention for `NameOfTheConcept_d` and `Name_of_a_class_d`.

`Arrangement_2` is part of the CGAL package with the same name [WFZH08] [WFZH07]. It allows, for instance, the construction of planar arrangements using a sweep-line algorithm (as described in Section 3.1.2) or incremental construction. The package also contains manipulation methods like insertion and deletion, and a method to compute the overlay of two arrangements (Section 4.2.4). Its template argument `ArrangementTraits_2` defines the relevant geometric types (`Curve_2` for curves/segments, `X_monotone_curve_2` for x-monotone curves/segments, `Point_2` for points), and operations on them, which are needed for the construction and maintenance of arrangements. Essentially, these operations are the geometric primitives needed by the generic sweep-line algorithm, as listed on page 97.

`Curved_kernel_via_analysis_2` [BE08] implements those types and operations on these types, again in a generic way; the concrete realization depends on the template argument `CurveAnalysisTraits_2`; a model of this concepts is required to provide a curve type and must be able to analyze curves of this type, and pairs of them, as defined in Section 3.1.4.[30] With such analyses at hand, the realization of the operations required by the `ArrangementTraits_2` concept is done as described on page 105.

`Algebraic_kernel_2` [EK08b] contains, as its main ingredient for our purpose, the curve analysis and curve pair analysis methods for arbitrary algebraic curves, as described in Sections 3.2 and 3.3, with the modifications described in Section 4.1. Moreover, the class is a model of the `AlgebraicKernel_2` concept [BHKT07], which encapsulates basic functionality for polynomials in two variables. The template argument `AlgebraicKernel_1` provides basic operations for univariate polynomials needed during the analysis and also implicitly determines the coefficient type of the polynomials.

Regarding the software part, the `Algebraic_kernel_2` class is the main contribution of the author of this thesis. Therefore, its internals will be specified in more detail in Section 4.2.2.

`Algebraic_kernel_1` [HL07] implements basic functionality for univariate polynomials (Section 2.4) and algebraic numbers (Section 2.5), such as square-free factorization, real root solving, and comparison of roots. One typically uses a coefficient type that models $\mathbb{Z}$, for instance, `leda::integer`[31] or `CORE::BigInt` [KLPY99]. However, other types for computationally more demanding domains are possible. We discuss one example in Section 5.2, where we consider arrangements of rotated algebraic curves.

The hierarchical structure of the presented class design allows the easy exchange of data types or complete layers while leaving the other layers intact. We have already mentioned that coefficient types beyond integers can be handled by the algebraic kernels. As another example, replacing the refinement method of algebraic numbers (currently, we use quadratic interval refinement as explained in Section 2.5.1) is simply the task of replacing the `Algebraic_kernel_1` model (or more precisely, just a functor therein). Even a more efficient method of analyzing curves and curve pairs would profit from the design, since the translation into geometric predicates, as done by the `Curved_kernel_via_analysis_2`, would remain valid as long as the new improved analyses are provided by a model of the `CurveAnalysisTraits_2` concept.

[30] The concept does not require that the curve type models algebraic curves all requirements can be reformulated using non-algebraic terms. However, since algebraic curves are the only instantiation so far, we skip a detailed discussion of this aspect.

[31] http://www.algorithmic-solutions.com/leda/

4.2.2. A closer look at the `Algebraic_kernel_2` class

We give some details about the `Algebraic_kernel_2` class, that is, about the requirements of the associated concept `AlgebraicKernel_2` and the implementation. Compared to the version discussed in the technical report [EK08b], the interface to the user has esentially remained unchanged, but the internal design has changed. Despite our goal of illuminating the class, we aim for a concise description; therefore, this treatment should not be understood as a manual for the `Algebraic_kernel_2` class.

The `AlgebraicKernel_2` concept A model for the concept `AlgebraicKernel_2` has to provide the following types and functors:[32]

- `Coefficient`: A number type that models the scalar coefficients of bivariate polynomials.
- `Polynomial_2`: A type that models bivariate polynomials over the scalar type `Coefficient`.
- `Algebraic_real_2`: A type that models points with algebraic coordinates.
- `Solve_2`: A functor that computes for two bivariate polynomials f and g a list of solutions for the system $(f = 0, g = 0)$ (in geometric language: a list of intersection points of $V(f)$ and $V(g)$).
- `Is_square_free_2, Make_square_free_2, Is_coprime_2, Make_coprime_2, Square_free_factorization_2`: Functors for checking for squarefreeness, for checking for coprimality, for decomposing two polynomials into a common part, and two coprime polynomials. and for computing the square-free part, for computing the square-free factorization.
- `Sign_at_2`: A functor to compute the sign of $f(\alpha)$, where f is of type `Polynomial_2`, and α is of type `Algebraic_real_2`.
- `Get_x_2, Get_y_2, Approximate_absolute_x_2, Approximate_absolute_y_2, Approximate_relative_x_2, Approximate_relative_y_2`: Functors to obtain the x- or y-coordinate of a point exactly, and to get a lower or upper bound of the x- or y-coordinates with respect to a certain absolute or relative precision.
- `Compare_xy_2, Compare_x_2, Compare_y_2`: Functors to compare two points (lexicographically), and to compare their x- or y-coordinates.

Our model `Algebraic_kernel_2` additionally contains models for two concepts which are required by the `CurveAnalysisTraits_2` concept. The concept `CurveAnalysis_2` reflects exactly what we have defined in Definition 3.1.7:

- Constructor: A curve analysis must be constructible from a bivariate polynomial.
- `Status_line_1`: A type that models f-stacks. It must be a model of the concept `CurveAnalysis_2::StatusLine_1`. Without giving formal details, this concept reflects the content of an f-stack as defined in Definition 3.1.6. Thus, it has methods to ask whether there is a vertical line, to get the fiber points at the f-stack as objects of type `Algebraic_real_2`, to obtain the branch numbers of each point, and to obtain the number of vertically asymptotic arcs on either side.
- `number_of_status_lines_with_event()`: A function that returns the number of critical x-coordinates of the curve.
- `status_line_at_event(i)`: Returns the `Status_line_1`-object for the i-th critical x-coordinate.

[32] A functor is an object that defines `operator()`, thus, it is something like an object that can be used as a function.

- `status_line_at_interval(i)`: Returns the `Status_line_1`-object for the i-th intermediate value.
- `status_line_at_exact_x(x)`: Returns the `Status_line_1`-object at the x-coordinate `x`.

Similarly, the `CurvePairAnalysis_2` concept reflects Definition 3.1.10:

- Constructor: A curve pair analysis must be constructible from a pair of bivariate polynomials.
- `curve_analysis(c)`: For $c \in \{1, 2\}$, returns the `Curve_analysis_2` object for the c-th curve of the curve pair.
- `Status_line_1`: A type that models fg-stacks. It must be a model of the concept `CurvePairAnalysis_2::StatusLine_1`. Without giving formal details, this concept reflects the content of a fg-stack as defined in Definition 3.1.9. Thus it has methods to ask whether the i-th fiber point is a point of $V(f)$, of $V(g)$, or of both, and to get the intersection multiplicity of intersection points (if the intersection is in the interior of segments).
- `number_of_status_lines_with_event()`: A function that returns the number of critical x-coordinates of the curve pair.
- `event_of_curve_analysis(i,c)`: Returns an integer $j \geq -1$ that denotes that the i-th critical x-coordinate of the curve pair is the j-th critical x-coordinate of the curve c; $j = -1$ denotes that the x-coordinate is not critical for the single curve.
- `status_line_at_event(i)`: Returns the `Status_line_1`-object for the i-th critical x-coordinate.
- `status_line_at_interval(i)`: Returns the `Status_line_1`-object for the i-th intermediate value.
- `status_line_at_exact_x(x)`: Returns the `Status_line_1`-object at the x-coordinate `x`.

Implementation: The core of the `Algebraic_kernel_2` is formed by the two classes `Curve_analysis_2` and `Curve_pair_analysis_2`. Their realization is based on the algorithms presented in this thesis. Concerning the storage of data, both for curves and curve paris, the critical x-coordinates (together with additional information (e.g., their multiplicity) are stored in an `std::vector`. Other `std::vector`s store the (rational) intermediate values and the principal Sturm-Habicht coefficients, or the principal subresultant coefficients, according to the case.

The classes differ slightly in how they store their stacks: `Curve_analysis_2` maintains an `std::map` that maps algebraic numbers (of type `Algebraic_real_1`) to f-stacks (of type `Status_line_1`. If the f-stack for some α has not been constructed yet, it is created and stored in the map. In this way, the curve analysis caches all f-stacks that have been constructed so far.

In the `Curve_pair_analysis_2`, there are only vectors that store an fg-stack for each critical and intermediate x-coordinate. If one asks for an fg-stack at some α where no fg-stack exists, the object determines which interval between two critical x-coordinates contains α and returns the fg-stack of the appropriate intermediate position. Note that the only information in the fg-stack is the vertical ordering of the points, which does not change inside an intermediate interval.

Both `Curve_analysis_2` and `Curve_pair_analysis_2` use a *lazy evaluation* technique. This means that all operations of the analysis are delayed as much as possible. The computation of the critical x-coordinates, the principal Sturm-Habicht coefficients, and the f-stacks are triggered only on demand. To give an example, consider the polynomial $f = (x^2 + y^2 - 1)((x+1)^2 + 4(y-1)^2 - 4)$ (Figure 4.6), and the following code.

```
Curve_analysis_2 ca(f);
int n = ca.number_of_status_lines_with_event();
for(int i=0;i<n;i++) {
  ca.status_line_at_event(i);
}
```

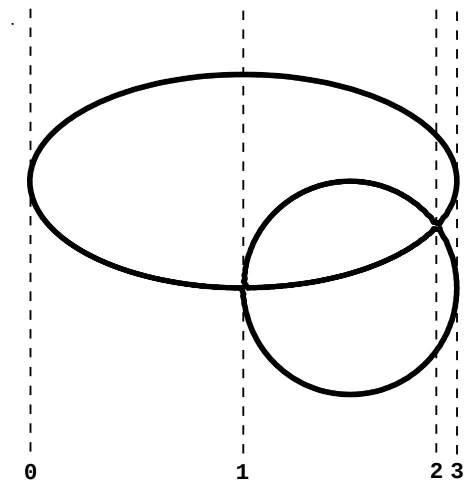


Figure 4.6. The curve $V(f)$, and its critical x-coordinates, numbered starting with 0.

In the first line, the curve analysis object is created. The only operation that is triggered is the storage of f as defining polynomial within the curve analysis object. In the second line, we ask for the number of critical x-coordinates. This triggers the computation of the resultant and the isolation of its real roots (and the computation of their multiplicities). For $i = 0$, we encounter a critical x-coordinate that is a simple resultant root. Thus, the simple method of Section 4.1.3 is applied. For $i = 1$, we ask for the status line at the second critical x-coordinate. This is not a simple root of the resultant, so we require the principal Sturm-Habicht coefficients to compute m and k for this step, in order to apply the m-k-bitstream-Descartes method. For $i = 3$, the m-k-bitstream-Descartes method fails. Therefore, the algorithm switches to a sheared system for the analysis in this situation.

The previous example demonstrated the advantage of the lazy evaluation strategy: if the f-stack at the rightmost critical x-coordinate is never queried, the curve analysis gets through without shearing the curve, although it is not in generic position. This is especially useful if one is only interested in certain parts of the curve, for instance, when computing the arrangement inside a predefined box. Also, the analogous technique for curve pairs (which is also implemented) is useful in the `Intersections` predicate: for that, one only considers the fg-stack in the common x-range of the involved segments.

In this context, we mention a further optimization step that is implemented as an optional feature. In CGAL, we currently use the algorithm by Ducos [Duc00] for computing the subresultants; If only the resultant is required, faster methods can be applied. For instance, the computer algebra system MAPLE (Version 11) uses modular methods for computing the resultant of bivariate polynomials with high degrees (see MAPLE's documentation of the `resultant` function). Indeed, this method is much faster than Ducos' subresultant algorithm. Also, an improved resultant algorithm based on interpolation has recently been implemented in CGAL by Michael Hemmer.

These faster resultant methods lead to the following strategy (which was already implicitly assumed in the example above). When computing the critical x-coordinate, we compute the resultant by a (relatively) fast method. As long as f-stacks of simple resultant roots are considered, there is no need to compute the principal Sturm-Habicht coefficients because the improved methods for simple resultant roots, presented in Section 4.1.3, do not require them. In particular, for curves only with simple resultant roots, we avoid computing them completely. However, if the Sturm-Habicht sequence must be computed, the resultant is computed once more as a by-product, so the initial resultant computation is a pure overhead.

We call this strategy the *resultant-first* strategy. Whether it should be used or not in the algorithm clearly depends on the expected input. If one considers many curves without singular points, it is likely that Sturm-Habicht computations can be avoided, so it is recommended to apply the strategy. It might also be useful if only small x-ranges of each curve are considered. However, if the input curves have singularities, and their complete analysis is computed, the resultant-first strategy will worsen the overall performance.

The `Algebraic_kernel_2` contains as its main data structure two caches that store all curve analyses (`curve_cache`) and all curve pair analyses (`curve_pair_cache`) that have so far been computed. For this purpose, the kernel contains functors `Construct_curve_2` and `Construct_curve_pair_2` which search for the desired curve or curve pair analysis in the corresponding cache and trigger the analysis only if it was not found. To avoid unnecessary analyses, all curves are first canonicalized, that is, their defining polynomials are made primitive with respect to their scalar coefficients and are forced to have a positive leading coefficient.

Most of the operations of `Algebraic_kernel_2` can be realized directly by considering the curve analysis or curve pair analysis of the curves involved. For instance, `Solve_2` for two polynomials f and g is realized in the following way. Construct the curve pair analysis of $V(f)$ and $V(g)$ (using the cache), iterate through the fg-stacks and return all intersection points. As another example, we consider `Sign_at_2` for $f \in \mathbb{Z}[x, y]$ and $p \in \mathbb{R}^2$. We first check whether $f(p) = 0$. For that, note that p is represented by a triple (α, f, J), where α is the x-coordinate of p, and J is an isolating for $g(\alpha, y)$ (recall the representation of points from Section 3.1.3). We construct the curve pair of $V(f)$ and $V(g)$ and check whether p is among the intersection points. If so, we return 0 as sign. Otherwise, we approximate p by a box and evaluate f at this box using interval arithmetic, until 0 is not contained in the resulting interval, so that the sign of $f(p)$ can be determined.

The only operations of `Algebraic_kernel_2` that do not follow (quite) directly from the analyses are `Get_y_2` and `Compare_y_2`. The former method returns the y-coordinate of a point in an isolating interval representation, which is not available directly by our representation (it is available only as an isolating interval of a polynomial $f(\alpha, y)$ with algebraic α). The latter operation requires checking y-coordinates for equality, which in turn requires an isolating interval representation if the points are at different x-coordinates. Thus, for a point p represented by (α, f, I), we need to compute an isolating interval representation (g, J).

This is done as follows: Assume that $V(f)$ has no vertical line components, otherwise, pass to its primitive part. We look up the branch numbers of p in the curve $V(f)$. If they do not sum to 2, p is singular, and its y-coordinate is a root of $R := \mathrm{res}_x(f, \frac{\partial f}{\partial y})$. Otherwise, p is still an intersection point of $V(f)$ and $V(R)$, where R is the defining equation of α (note that $V(R)$ is a union of vertical lines, one for each root of $R \in \mathbb{Z}[x]$). Thus, the y-coordinate is a root of $\mathrm{res}_x(f, R)$. In either case, we found a defining polynomial g for the y-coordinate of p.

To find the isolating interval, we isolate the real roots of g and refine the y-coordinate of p until its interval only overlaps with one of the isolating intervals of g. Then the corresponding interval can be chosen as the isolating interval for the y-coordinate.

We recommend using the methods `Get_y_2` and `Compare_y_2` carefully, because the representation of y-coordinates in an isolating interval representation does not really fit our projection-based framework. Thus, these methods cause a serious overhead due to additional symbolic computations – in particular, the computation of $\mathrm{res}_x(f, R)$ is extremely expensive, because R is itself a resultant polynomial of roughly quadratic degree compared to $\deg f$. Fortunately, computing arrangements of arbitrary algebraic curves is possible without the usage of the functions `Get_y_2` and `Compare_y_2`.

As a final remark, we do not claim that our implementation of `Solve_2` is optimal for the problem of solving a bivariate system in general: For finding the intersection points of two algebraic curves, it is certainly not the best solution to analyze each single curve first and perform a curve pair analysis afterwards. However, the additional cost of performing a curve analysis for each occurring curve might amortize when a curve is intersected with many other curves. This is particularly the case when arrangements of algebraic curves are computed.

4.2.3. Filtered kernels

Despite all our optimization efforts, computing a curve pair analysis remains a time-consuming task, that our `Algebraic_kernel_2` engages in extensively. In some situations, a simpler method might also be successful. For instance, reconsider the `Sign_at_2` functor, where we compute $\mathrm{sign}(f(p))$ for $f \in \mathbb{Z}[x, y]$ and $p = (\alpha, g, I)$. Instead of starting with computing the curve pair analysis of $V(f)$ and $V(g)$ to check for equality, we could first approximate p by a box B (whose side length is determined by a fixed threshold), and evaluate $\square f(B)$. If 0 is not contained, we have computed the sign without computing a curve pair analysis (of course, if $f(\alpha) = 0$, there is no way to prevent its computation, no matter what threshold is chosen). The same idea can be applied to the `Compare_xy_2` and the `Compare_y_2` predicates. We provide a class `Filtered_algebraic_kernel_2` that realizes these filtered operations (currently with a threshold of $\frac{1}{100}$).

The same idea is also applicable for the `Curved_kernel_via_analysis_2`. One of its most important functionalities is the `Intersections` operation, which computes all intersection points in the interior of two segments. The hope is to quickly decide cases where the segments do not intersect at all, without querying the curve pair analysis of the curves involved. For that, each segment is approximated by a region that contains the whole segment, and the empty set is returned as the set of intersection points if the regions of two segments are disjoint. This avoids the curve pair analysis, at least if two segments are well separated from each other.

In our current preliminary implementation, we take the simplest possible region to bound a segment: an axis aligned bounding box. For this purpose, we have to identify the y-extremal points of the curve on the segment, which requires an additional curve pair analysis of $V(f)$ and $V(\frac{\partial f}{\partial x})$. This increases the running time, but has to be done only once per curve, and should amortize at least when considering a lot of curve pairs. Some experimental studies are reported in [Ker08]. Besides simple optimizations as described in Figure 4.7, improvements are certainly achievable by closer approximations of segments.

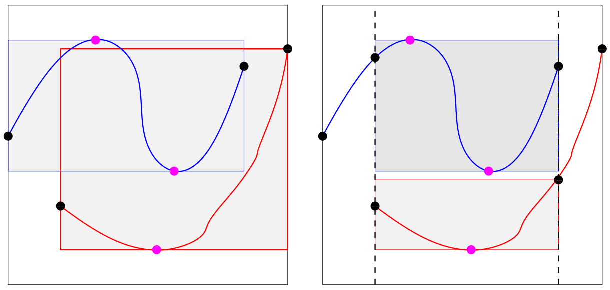


Figure 4.7. Consider the red and blue segments on the left. Their bounding boxes intersect each other, therefore, the filtered intersection predicate does not give an answer. However, both segments can be trimmed to the common x-range for the intersection test (on the right). After that, their bounding boxes are disjoint, thus, the segments do not intersect.

4.2.4. Overlays of arrangements

The ability to compute overlays of arrangements is not quite a feature of our implementation, but rather a consequence of it. Still, we will need overlays in several applications in the subsequent chapters. Let us first define what we mean by an overlay.

Definition 4.2.1 (overlay). Let the arrangement $\mathcal{A}_1$ be induced by the segments $s_1^{(1)}, \ldots, s_{m_1}^{(1)}$, and the arrangement $\mathcal{A}_2$ be induced by the segments $s_1^{(2)}, \ldots, s_{m_2}^{(2)}$. The *overlay* $\mathcal{O}$ of $\mathcal{A}_1$ and $\mathcal{A}_2$ is the arrangement induced by $s_1^{(1)}, \ldots, s_{m_1}^{(1)}, s_1^{(2)}, \ldots, s_{m_2}^{(2)}$.

Observe that each cell of the overlay can be expressed by the intersection of a cell of $\mathcal{A}_1$ with a cell of $\mathcal{A}_2$. The sweep-line algorithm presented in [dBvKOS00] to compute overlays in the linear case generalizes to curved segments, and the same geometric predicates as for the arrangement computation are required. The `Arrangement_2` package of CGAL provides a generic algorithm for computing overlays, relying on a model of the `ArrangementTraits_2` concept [WFZH08]. Therefore, our software framework can be used to compute overlays of arrangements induced by algebraic segments.

A cell of a CGAL arrangement can store additional data associated with the cell. The question is what data should be assigned to the cells of the overlay. An `OverlayTraits_2` concept that requires functions for this purposes is specified; it is passed as template parameter to the overlay computation. A model of this concept defines a function that takes a cell c_1 of $\mathcal{A}_1$, and a cell c_2 of $\mathcal{A}_2$ as arguments, and assigns the correct data to the cell of the overlay that is the intersection of c_1 and c_2. Using that mechanism, we can also assume w. l. o. g. that each cell in the overlay stores the cells c_1 of $\mathcal{A}_1$ and c_2 of $\mathcal{A}_2$ that it originates from.

Summary

The layered approach for computing arrangements of algebraic curves described in Chapter 3 is reflected in our software design, which has been implemented in the context of the CGAL library. In particular, the curve analysis and curve pair analysis form the two workhorses of the class `Algebraic_kernel_2`, which realizes basic algebraic operations on bivariate polynomials. Our flexible software design, based on template code, allows us to apply the code also to the arrangement of rotated curves, and arrangements on parametrizable surfaces, as we will describe in Chapter 5.

4.3. Experiments

All our experiments are performed on an Intel Core 2 dual core, clocked with 2 Ghz each, with 3 GB of RAM and 6 MB of cache. The workstation is running under Debian GNU/Linux 5.0.2 (lenny), with kernel version 2.6. Our software currently does not benefit from having several processors, although many steps of the algorithm are well-suited for parallel computation.

We compiled our programs with the GNU g++ compiler, version 4.1. For the underlying CGAL library, we used public release 3.4.CGAL supports several third-party libraries; we used Boost 1.39,[33] GMP 4.2,[34] and LEDA 6.2.1. We compiled our programs with the compiler option "-O3 -DNDEBUG", and also used the resultant-first strategy (Section 4.2.2) in all instances. We tested both with the number types provided by CORE (which is part of the CGAL release) and by LEDA. Because the CORE types (which are based on GMP) showed a better general behavior, we only display the results obtained using CORE in the following experiments.

4.3.1. The case of one curve

There is an enormous diversity of algebraic curves. Therefore, it is already difficult to define a set of meaningful test instances. We mention several parameters that have an influence on the running time of our algorithm:

- The degree and coefficient bitsize of the curve.
- The number of critical positions (because more critical values lead to more lifting steps).
- The presence of singularities (because they trigger the computation of subresultants and computing stacks at singular positions requires more symbolic computations).
- The presence of covertical critical points (because they trigger a shear) and of vertical asymptotic arcs (because they might trigger a shear).

We consider five different families of curves (see also Figure 4.8):

1. Fix two integers n and c, and choose f as a (dense) polynomial of degree n, with each coefficient a random integer in the range $[-2^{c-1}+1, 2^{c-1}]$. This is the common way of producing "random curves". Such curves are the simplest instances for given

[33] `http://www.boost.org/`
[34] `http://gmplib.org/`

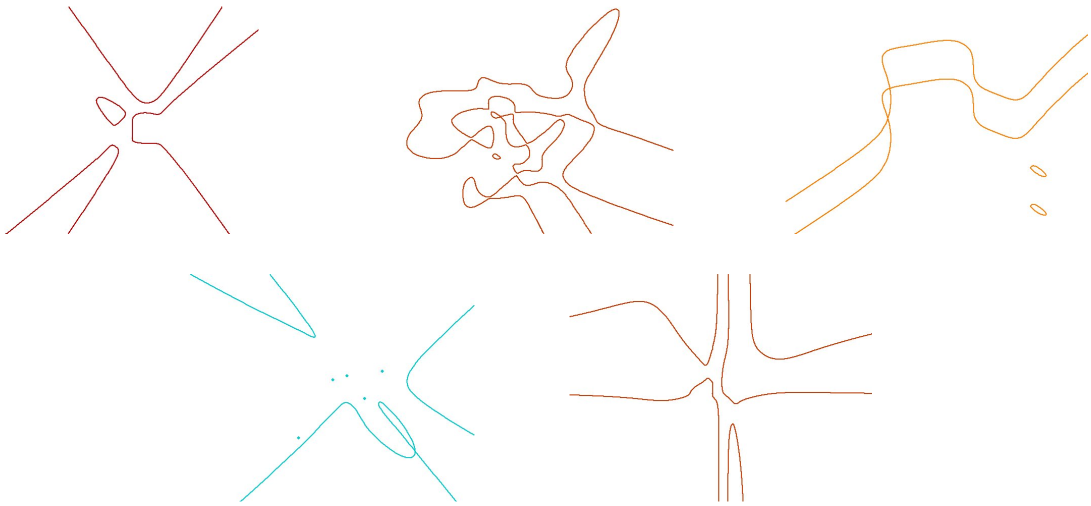


Figure 4.8. Plots of curves of type rand(11,50), inter(10), trans(7,25) (upper row, left to right) , res(3,3,8), and param(6,16) (lower row, left to right)

degree and bitsize, because in general, they do not have singular points, are in generic position, and have few real segments. We will denote such curves by rand(n,c).

2. Fix a degree n, and a $2n \times 2n$ integer grid in $\mathbb{R}^2$ with points (i, j), with $-n+1 \leq i, j \leq n$. Choose $\frac{1}{2}n(n+3)$ grid points randomly, and construct a curve of degree n that goes through all these grid points. The curve obtained is still regular in general. Due to the interpolation, it consists of many more (real) segments, and also the coefficient size increases with the degree. We denote such curves by inter(n).
   Why do we choose exactly $\frac{1}{2}n(n+3)$ interpolation points? Note that $\frac{1}{2}n(n+3) = \binom{n+2}{2} - 1$ and that a general polynomial of degree n has exactly $\binom{n+2}{2}$ coefficients. Forcing a point to lie on the curve is a linear condition on the coefficients; imposing $\binom{n+2}{2} - 1$ such conditions, the curve still has (at least) one degree of freedom, and therefore, there exists a non-zero polynomial that satisfies all these equations. This shows that an interpolated curve of this form always exists.
3. For n and c, choose a $g(x, y)$ as in 1, and consider the curve defined by $f(x, y) = g(x, y)g(x, y + 1)$. The effect is that all critical points of $V(g)$ appear in covertical pairs, so $V(f)$ is not in generic position. We denote such curves by trans(n,c). Note that f is of magnitude $(2n, 2c)$.
4. Fix degrees n_1, n_2, and a bitsize c, and construct two trivariate polynomials $F_1, F_2 \in \mathbb{Z}[x, y, z]$ with $\deg F_i = n_i$, analogous to 1. Then, define $f(x, y) := \mathrm{res}_z(F_1, F_2)$. Geometrically, this is the intersection curve of two algebraic surfaces, projected onto the xy-plane. The polynomial is of degree $n_1 \cdot n_2$ and its coefficients are upper bounded by $(c+\log(n_1+n_2))(n_1+n_2)+2\log(2(n_1+n_2)+1)$ [BPR06, Prop.8.12]. In general, f contains singular points [McC99] but it is in generic position. We denote such curves by res(n_1,n_2,c).
5. For n and c, construct a polynomial $F \in \mathbb{Z}[x, y, z]$ as in 4, and homogenize it, that is, multiply each monomial by some w^k (with w as a new indeterminate), such that each

monomial has the same degree. The homogenized polynomial $\hat{F}$ is then evaluated at

$$\begin{pmatrix} 2u(1-v^2) \\ 4uv \\ (1-u^2)(1+v^2) \\ (1+u^2)(1+v^2) \end{pmatrix}$$

with u, v as new indeterminates. The result is a polynomial $f \in \mathbb{Z}[u, v]$. Geometrically, this curve expresses the intersection of the surface defined by F with the unit sphere, in a certain parameter space of the unit sphere. Such curves in general have no singularities, but many (horizontal and vertical) asymptotic arcs. We denote such curves by param(n,c).

We have written a test program based on the curve analysis algorithm from the previous chapter and the optimizations presented in this chapter. It computes all f-stacks at critical x-coordinates, and for intermediate stacks in between, and refines the y-interval of each stack point to a width of less than $\frac{1}{100}$. We refer to this program by the name `CA`.[35]

We compared `CA` with two other programs that compute a similar output using distinct methods. Brown's `cad2d` is an optimized version of `QEPCAD-B`[36] (Version 1.50) for computing a cylindrical algebraic decomposition (*cad*) in the plane. It is written in C++. Its advantage over the more general `QEPCAD-B` is that it uses floating-point methods in the lifting step to simplify calculations in favorable situations. Brown describes such optimizations in [Bro02]. `cad2d` is able to produce *cad*s for an arbitrary number of curves, but we restrict ourselved to one curve for the comparison with our method.

We run `cad2d` with support of the Singular library[37] (version 3.1.0) and SACLIB (version 2.2.0). By default, `cad2d` does not compute the adjacencies of the *cad*. This computation, however, can be forced by a subsequent call of the `closure2d` command, which computes adjacencies as the first step.[38] Also, we called `cad2d` with option `+N10000000` since it runs out of memory with the default settings for some instances.

Additionally, we tested the MAPLE implementation of `isotop` [CLP+09], which is available online.[39] It is based on the MAPLE package `RS` to solve non-linear systems of equations using the rational univariate representation (RUR). The implementation of `isotop` comes with two versions of computing isolating boxes from the RUR, one fast but unreliably, using `RS`, and one slower but stable "self-made" version (controlled by the flag `Compute2DboxeswithRS`). Following the advice of the authors,[40] we tried both options and observed a small speed-up when using `RS` (<5%), but the algorithm sometimes seems to enter an infinite loop because it does not finish for hours. For that reason, and because the version not using `RS` in this step is always exact, all our runtimes relate to the version with the flag set to *false*.

There are MAPLE implementations of other methods to compute the topology available: `top` by Gonzalez-Vega and Necula [GVN02], and `insulate` [SW05] by Seidel and Wolpert. Previous test runs [Ker06] [EKW07] [CLP+09] have already shown that they

[35] The "traditional" name `AlciX`, as used in [Ker06] [EKW07] [EK08a] [CLP+09] stood for "**Al**gebraic **c**urves **i**n E**X**ACUS", but since the code has moved from the Exacus library into an experimental Cgal package, the original name no longer makes sense.

[36] `http://www.cs.usna.edu/~qepcad/B/QEPCAD.html`

[37] `http://www.singular.uni-kl.de/`

[38] We thank Christopher Brown for this advice.

[39] `http://webloria.loria.fr/equipes/vegas/isotop/`

[40] We thank Marc Pouget for his comments.

Curve	CA	res	solve	lift	cad2d	isotop
rand(8,50)	0.08	41%	12%	44%	0.14	0.94
rand(11,50)	0.29	57%	12%	29%	0.59	5.92
rand(14,50)	0.95	63%	13%	22%	2.27	27.7
rand(17,50)	2.98	64%	17%	17%	4.42	85.64
rand(20,50)	7.33	72%	17%	10%	13.29	*300.57
rand(23,50)	17.82	74%	17%	7%	19.13	–
rand(26,50)	36.19	80%	14%	5%	35.49	–
rand(10,64)	0.20	55%	12%	31%	0.42	4.65
rand(10,256)	0.73	76%	10%	12%	2.71	43.52
rand(10,1024)	4.80	90%	4%	4%	38.26	623.47
rand(10,4096)	36.26	97%	1%	0%	–	–
rand(10,16384)	224.20	96%	1%	1%	–	–

Table 4.1. Benchmark results for random curves (timings in seconds). `isotop` returned with an error for one of the instances of rand(20,50) – the running time is the mean of the other four instances.

perform, in general, worse than `CA` and `isotop`, thus, we decided to leave them out of our experiments.

Besides the total running time of `CA`, `isotop`, and `cad2d`, we also measured the timings for several substeps of `CA`. We considered the time to compute (sub)resultants and Sturm-Habicht sequences of the curve and of all sheared curves that must be considered during the algorithm (the results are in the colum "res") and also the time to isolate the roots of resultants, which means computing its square-free factorization and applying the root isolation algorithm to it (column "solve"). Also, we give the time to construct the f-stacks at critical x-coordinates. This contains the symbolic precomputation of the values m and k per stack, and the application of the m-k-bitstream-Descartes method, for the original and all sheared curves (column "lift"). The timings are given by their percentage with respect to the total running time. We also measured the time for the intermediate stacks, but this was less than 1% in every example considered. For all test instances, we created five examples (and the same set of curves was used for each program). The timings listed are the mean of those test runs.

Table 4.1 shows the results for random curves. We first observe that the symbolic computations in our curve analysis become the crucial operation when the degree increases, and especially when the coefficient size increases. We can observe that `CA` and `cad2d` are roughly equally fast for increasing degrees, and both are much faster than `isotop` for moderate bitsizes. The reason is probably that both `CA` and `cad2d` are optimized for non-singular curves (for `CA`, we use the resultant-first strategy and the optimization for simple resultant roots as explained in Section 4.1.3), whereas `isotop` solves the system $f = \frac{\partial f}{\partial y} = 0$ symbolically as its first step for any (singular or regular) instance.

For increasing coefficient sizes, we see that `CA` is much faster than both `cad2d` and `isotop`. Moreover, the latter two approches do not even compute a result for big coefficient sizes: `cad2d` reports "Prime list exhausted", denoting that the interally stored list of primes was not sufficient to perform a modular computation during the algorithm. `isotop` prints

Curve	bitsize	CA	res	solve	lift	cad2d	isotop
inter(6)	64	0.09	12%	7%	72%	0.10	0.43
inter(8)	153	0.34	25%	15%	55%	1.05	4.58
inter(10)	312	1.44	47%	15%	35%	8.82	50.69
inter(12)	527	6.74	60%	17%	21%	78.34	485.75
inter(14)	847	28.98	71%	17%	10%	471.46	–
inter(16)	1252	104.66	75%	17%	7%	–	–

Table 4.2. Benchmark result for randomly interpolated curves.

Curve	CA	res	solve	lift	cad2d	isotop
trans(5,25)	1.22	46%	27%	26%	26.12	1.07
trans(6,25)	3.77	58%	26%	14%	54.12	2.44
trans(7,25)	10.70	69%	21%	8%	190.21	*6.49
trans(8,25)	27.16	76%	17%	5%	–	14.23
trans(9,25)	65.12	82%	13%	3%	–	35.65
trans(10,25)	147.19	86%	10%	2%	–	69.74

Table 4.3. Benchmark results for random non-generic curves. For one of the instances in trans(7,25), `isotop` returned an error.

out a MAPLE error message within the Groebner basis computation (after more than one hour of computation). In `cad2d`, we also observed some "outliers" for feasible examples, where the performance was more than 10 times worse than in "typical" cases.

We cannot give an algorithmic argument for the success of `CA` – we believe that the stable and efficient number type support in CGAL for arbitrarily-sized integers leads to these good results, whereas the other algorithms just might not have been designed for such cases.[41]

The comparison results for randomly interpolated curves (Table 4.2) give a similar result – for increasing degree (which implies an increasing bitsize), `CA` performs much better than `isotop` and `cad2d`. It is still remarkable that the difference between the methods is already observable for instances with relatively moderate bitsizes. It can be concluded that the case of many critical points is handled better by `CA` than by the other two approaches.

For curves of type trans(n,c), we observe that `isotop` is slighly faster than `CA`, roughly by a factor of 2. This result is expected, because `CA` has to perform a shear for such curves and performs the analysis in a different coordinate system. Moreover, the curve in general contains self-intersections. In contrast, `isotop` is unaffected by the coverticalness of critical points. `cad2d` is much slower in such instances. The reason is that the optimized lifting procedure is not applied in the presence of covertical critical points.[42] Thus, the

[41] Marc Pouget (one of the authors of `isotop`) mentioned in private communication that his group is aware of the problem, and they are planning to optimize their algorithm for big coefficient sizes in a later release.

[42] `cad2d` factorizes the input as its first step and thereby obtains two polynomials for which the optimization would be applicable. However, according to [Bro02], the purely symbolic method is used if the

Curve	CA	res	solve	lift	cad2d	isotop
res(2,3,8)	0.05	23%	32%	39%	0.06	0.21
res(3,3,8)	0.30	38%	30%	29%	2.23	0.89
res(3,4,8)	1.58	64%	27%	8%	87.94	3.15
res(3,5,8)	8.15	76%	18%	4%	–	9.49
res(4,5,8)	77.84	89%	9%	1%	–	54.81
res(5,5,8)	481.60	93%	5%	0%	–	275.79

Table 4.4. Benchmark results for random singular curves.

Curve	CA	res	solve	lift	cad2d	isotop*
param(2,16)	0.03	10%	19%	63%	0.04	0.15
param(4,16)	0.32	37%	25%	36%	0.17	1.79
param(6,16)	2.57	65%	18%	16%	1.17	30.26
param(8,16)	7.96	27%	37%	35%	9.85	403.69
param(10,16)	27.18	39%	36%	23%	30.68	–

Table 4.5. Benchmark results for random curves induced by the intersection of a random surface with the unit sphere. `isotop` returned an error for some instances; we report the average time for the working examples.

method falls back on a completely symbolic method in this case. For degrees greater than 7, it either runs out of memory or it quits with the error message "Prime list exhausted".

For singular curves, we get similar results as for non-generic ones (Table 4.4). `isotop` and `CA` show a similar behavior, with an advantage to `isotop` for higher degrees. Again, the subresultant computation, triggered by the occurence of singular points, results in more symbolic computations in `CA` than in regular examples (note also that the relative amount of symbolic computation increases for higher degrees). Compared to that, `cad2d` is much slower, because its optimizations do not apply in the presence of singular points.

Table 4.5 shows the set of parameter curves for which `cad2d` and `CA` show similar performance; `isotop` is much slower than both (and returns an error in some instances). Once more, the reason lies in the optimized handling of simple resultant roots, both for `CA` and `cad2d`, because this optimizations also apply in the presence of vertically asymptotic arcs (Section 4.1.4).

As a result of our comparison, we can report that our implementation `CA` is currently the most stable approach among the three tested algorithms tested. It returns the result much faster than the others, especially for higher coefficient sizes, and in other cases, it is at least not too far away from the optimal choice. Compared to `cad2d`, this is not a surprising result, since `cad2d` only speeds up the *cad* computation for simple examples, whereas our approach uses fast adaptive-precision techniques in each instance, and never falls back on pure symbolic computations. We do *not* deduce from the results, however, that the projection approach used in `CA` is, in principle, superior to the subdivision approach used in `isotop`. A more mature implementation of their algorithm (and of the underlying MAPLE

x-coordinate is a root of more than one factor of the input.

package RS to solve non-linear systems) might arrive at similar, if not better, performance results. Moreover, pure subdivision algorithm that do not use symbolic computations at all [BCGY08] constitute a promising alternative. More experiments are certainly needed when mature implementations for such techniques are available.

4.3.2. The case of several curves

We turn to the arrangement computation of algebraic curves. Besides the influential factors mentioned above that affect the performance (such as degree, bitsize, presence of singularities, etc.), we have an additional parameter, namely the number of input curves. We split the total runtime into six independent quantities:

- **res**: Time for computing (sub)resultants and Sturm-Habicht sequences of one curve and of curve pairs.
- **solve**: Time for the square-free factorization of resultants and the isolation of the real roots, for both single curves and for curve pairs.
- **lift**: Time to produce f-stacks of curves and fg-stacks of curve pairs at critical positions.
- **inter**: Time to produce intermediate f-stacks of curves and intermediate fg-stacks of curve pairs.
- **compare**: Time to compare the x-coordinates of points during the sweep-line algorithm (this is called for whenever event points are inserted into the event queue).
- **decompose**: Time to compute the square-free part of input curves, and to decompose a curve pair into a common part and two coprime remainder curves. Note that all our input arrangements consist of square-free and pairwise coprime curves. Therefore, we can switch off this computation (and our implementation provides a compiler flag for that), but we decided to keep it in order to show its cost.

Recall that all our test for the curve analysis has been performed on five randomly generated curves of certain types. We compute the arrangement induced by those five curves together (Table 4.6).

We observe that the behavior for high degrees and high coefficient sizes is similar to the one-curve case: The symbolic computations become the dominant factor in the computations; note that not just the column labeled "res", but also the column "decompose" performs such symbolic computations.

In the column labeled "ca", we also denote the time to compute the curve analysis of the input instances. Clearly, this operation must be executed before the sweep starts in order to obtain x-monotone segments. As we have to perform five curve analyses, and up to $\binom{5}{2} = 10$ curve pair analyses, we expect the total running time to be roughly three times as big as the time for the curve analysis, assuming that a curve pair analysis is roughly as expensive as a curve analysis. We see that most examples match this rule of thumb, except the inter$(\cdot)$ arrangements which are much worse. This is because they are in some sense the worst possible instances for our arrangement algorithm: recall that they were created by interpolating points on a $2n \times 2n$ integer grid. Considering a curve pair, it is likely that intersections take place in grid points, and it may also happen that some intersections are covertical. Thus, the curve pair must be sheared. The effect is that grid points are mapped to grid points in the sheared system, and it is not unlikely that other intersection points become covertical, especially if the shear factor is small. Recall from Section 4.1.5 that we prefer to choose small shear factors in the beginning, thus, a lot of

Curves	#segs	#(V,E,F)	ca	total	res	solve	lift	inter	comp.	decomp.
rand(8,50)	56	(78,120,44)	0.34	**1.34**	43%	12%	10%	7%	1%	23%
rand(12,50)	98	(152,230,79)	2.26	**10.60**	41%	10%	6%	4%	2%	30%
rand(14,50)	98	(154,234,81)	4.63	**23.75**	42%	10%	4%	2%	2%	33%
rand(17,50)	149	(198,273,78)	14.72	**73.54**	43%	10%	3%	2%	2%	34%
rand(20,50)	102	(164,238,76)	36.31	**185.04**	47%	8%	1%	0%	2%	34%
rand(23,50)	153	(216, 297, 82)	86.89	**461.98**	46%	9%	1%	1%	2%	32%
rand(26,50)	142	(200, 282, 84)	182.01	**1032.48**	45%	8%	0%	1%	1%	36%
rand(10,64)	82	(122,194,74)	0.98	**4.50**	44%	12%	6%	5%	1%	25%
rand(10,256)	80	(124,180,57)	3.54	**13.90**	72%	9%	2%	2%	1%	8%
rand(10,1024)	86	(134,190,57)	23.88	**89.94**	87%	3%	1%	1%	1%	1%
rand(10,4096)	34	(66,114,49)	180.94	**688.98**	92%	1%	0%	0%	1%	0%
rand(10,16384)	88	(146,224,80)	1119.00	**4480.92**	85%	1%	0%	2%	2%	0%
inter(6)	248	(410,605,196)	0.32	**2.99**	15%	15%	42%	19%	1%	2%
inter(8)	506	(823,1175,353)	1.48	**22.87**	29%	21%	36%	7%	1%	1%
inter(10)	904	(1367,1864,498)	6.80	**167.23**	47%	17%	25%	4%	1%	0%
inter(12)	1546	(2223,2945,724)	33.05	**1753.54**	65%	13%	15%	1%	1%	0%
inter(14)	2286	(3096,3963,868)	143.67	**9285.85**	76%	9%	9%	1%	0%	0%
inter(16)	3228	(4275,5382,1108)	521.87	**65363.00**	78%	6%	10%	2%	0%	0%
trans(5,25)	218	(370,598,229)	5.88	**21.07**	36%	24%	33%	5%	0%	5%
trans(6,25)	200	(290,444,155)	19.84	**52.89**	53%	23%	20%	2%	0%	6%
trans(7,25)	226	(334,506,173)	55.10	**140.89**	61%	19%	17%	1%	0%	5%
trans(8,25)	188	(290,460,171)	139.65	**329.66**	69%	16%	12%	0%	0%	5%
trans(9,25)	310	(418,606,189)	337.42	**760.61**	74%	13%	11%	0%	0%	4%
trans(10,25)	220	(328,508,181)	768.99	**1610.92**	76%	11%	11%	0%	0%	4%
res(2,3,8)	70	(86,118,37)	0.18	**0.49**	30%	20%	17%	14%	0%	13%
res(3,3,8)	199	(232,313,96)	1.41	**3.66**	32%	19%	15%	8%	1%	16%
res(3,4,8)	126	(146,174,45)	7.57	**16.07**	49%	18%	5%	1%	0%	20%
res(3,5,8)	301	(366,461,116)	40.61	**75.65**	56%	15%	4%	1%	1%	15%
res(4,5,8)	287	(354,452,126)	383.30	**594.56**	70%	9%	1%	0%	1%	10%
res(5,5,8)	542	(588,713,155)	2399.05	**3223.24**	80%	7%	0%	0%	0%	7%
param(2,16)	58	(70,98,29)	0.13	**0.24**	13%	21%	38%	10%	0%	11%
param(4,16)	120	(148,200,54)	1.46	**3.34**	24%	18%	14%	5%	0%	35%
param(6,16)	130	(180,258,79)	12.24	**39.83**	29%	11%	4%	1%	0%	51%
param(8,16)	552	(726,932,210)	38.99	**228.39**	15%	14%	5%	1%	3%	51%
param(10,16)	552	(726,940,216)	132.88	**977.00**	17%	11%	2%	1%	5%	45%

Table 4.6. Benchmark results for various arrangments of 5 curves of a certain type.

#Curves	#segs	#(V,E,F)	ca	total	res	solve	lift	inter	comp.	decomp.	per vertex
10	116	(314,552,239)	0.63	**5.34**	40%	10%	4%	9%	5%	26%	17032
20	228	(1066,1976,911)	1.36	**21.15**	40%	11%	2%	7%	7%	28%	19847
30	292	(1782,3396,1615)	1.88	**44.56**	41%	10%	1%	5%	7%	30%	25009
40	504	(3718,7080,3363)	2.82	**82.54**	39%	10%	1%	5%	8%	29%	22201
50	684	(6462,12448,5987)	3.59	**135.94**	37%	10%	1%	5%	10%	28%	21037
60	684	(7554,14636,7083)	4.11	**177.04**	39%	10%	1%	5%	10%	29%	23436
70	896	(11504,22368,10865)	4.86	**253.95**	38%	10%	0%	5%	11%	28%	22075
80	948	(13084,25552,12469)	5.53	**318.27**	39%	10%	0%	4%	10%	29%	24325
90	1042	(17644,34570,16927)	6.14	**410.31**	38%	10%	0%	4%	11%	29%	23254
100	1090	(21088,41462,20375)	6.65	**509.34**	38%	10%	0%	4%	11%	29%	24153

Table 4.7. Benchmark results for m curves of type rand(8,50).

shear transformation will be necessary to put each curve pair into a generic position.

All examples in Table 4.6 are restricted to five curves. We fix a degree of 8 and a bit size of 50 and construct arrangements of m randomly generated curves (Table 4.7). The relative running times of the subroutines remain rather stable when the number of curves increases. Again, the symbolic computations contribute the major part of the running time. Also, the relative cost of the curve analyses becomes smaller, clearly because the number of curve pair analyses grows quadratically with the number of curves. Arrangements of considerable size arise from these instances; in the last column, we show how many microseconds are spent per output vertex on average. This number is basically constant. Indeed, this matches the complexity bound of the sweep-line algorithm, which is (up to a logarithmic factor) linear in the number of output nodes, assuming that all geometric primitives take a constant time to evaluate.

We also tested our implementation on arrangements of restricted classes of curves, which are available in the CGAL library. For circles and circular arcs, CGAL provides a model for the `ArrangementTraits_2` concept, called `Arr_circle_segment_traits_2`. We created m circles, each represented by a triple $(i, j, r) \in \{1, \ldots, m\}^3$ chosen uniformly at random, where (i, j) is the center of the circle and r is its radius (Table 4.8). Again, the distribution of the total time to the subroutines is more or less independent of the number of input circles. The influence of symbolic computations is smaller than for the previous instances, because circles are simpler to handle symbolically then curves of degree 8. Again, the costs per node are roughly constant. Observe that the last example produces an arrangement of more than two million cells.

We observe that the CGAL implementation is faster than our approach, roughly by a factor of 6 . This appears plausible because a specialized implementation for certain types of curves should perform better than a more general approach. We have not particularly optimized our code for the case of many curves with low degrees (we were more focussed on curves with higher degrees).

Additionally, traits classes are available in the CGAL library for (bounded) arcs of conics (`Arr_conic_traits_2`) and rational functions of the form $y = \frac{p(x)}{q(x)}$ with p, q univariate polynomials (`Arr_rational_arc_traits_2`). Both classes of curves can be handled with our implementation as well (for the latter, the defining polynomial of the curve is just

#Circles	#(V,E,F)	CGAL	CA	res	solve	lift	inter	comp.	decomp.	per vertex
100	(5459, 10726, 5269)	0.63	**4.44**	25%	7%	8%	25%	7%	13%	813
200	(24268, 48142, 23877)	3.00	**18.67**	21%	6%	9%	20%	12%	14%	769
300	(52594, 104597, 52005)	6.81	**40.11**	23%	6%	7%	18%	13%	15%	762
400	(85081, 169374, 84295)	11.39	**64.60**	22%	6%	8%	17%	14%	15%	759
500	(134984, 268973, 133991)	18.66	**101.96**	22%	6%	7%	16%	15%	15%	755
600	(192640, 384084, 191447)	27.21	**144.63**	22%	6%	7%	15%	16%	15%	750
700	(260519, 519651, 259134)	37.31	**196.53**	22%	6%	7%	14%	17%	15%	754
800	(338786, 675986, 337202)	50.31	**256.11**	22%	6%	7%	14%	17%	15%	755
900	(440708, 879619, 438914)	65.15	**331.82**	22%	6%	7%	13%	18%	15%	752
1000	(548041, 1094088, 546050)	82.01	**412.14**	22%	6%	7%	13%	18%	16%	752

Table 4.8. Benchmark results for m circles with center on an $m \times m$ grid and random (integer) radius between 1 and m.

#Ellipses	#(V,E,F)	CGAL	CA	res	solve	lift	inter	comp.	decomp.	per vertex
10	(66, 112, 49)	15.00	**0.12**	29%	3%	13%	23%	3%	13%	1818
30	(538, 1016, 480)	57.04	**0.56**	20%	9%	12%	32%	8%	9%	1040
50	(1556, 3012, 1458)	110.27	**1.49**	19%	8%	10%	27%	11%	15%	961
70	(2900, 5660, 2762)	172.17	**2.79**	18%	7%	8%	27%	13%	14%	962
90	(4656, 9132, 4478)	238.27	**4.43**	22%	8%	9%	24%	12%	14%	951
200	(23744, 47088, 23346)	816.71	**21.71**	20%	8%	7%	21%	18%	13%	914
300	(49898, 99196, 49301)	1514.33	**45.59**	20%	8%	6%	20%	20%	13%	913
400	(96502, 192204, 95704)	2620.06	**88.64**	20%	8%	6%	18%	22%	13%	918

Table 4.9. Benchmark results for m random ellipses.

$y \cdot q(x) - p(x))$. We compared our implementation with the conic traits class[43] by computing arrangements of random ellipses with 30-bit coefficients. We also compared it with rational curves, by fixing degree n and bitsize τ, choosing p and q as random univariate polynomials of magnitude (n, τ) (in an analogue way as for `rand`(n,τ)), and computing the arrangement of m such curves. Because this traits class supports unbounded segments, we simply take the arrangement induced by the complete curves.

We display the results in Tables 4.9 and 4.10. The first result is that our code is faster for ellipses than the conic-specific code. We concur with Ron Wein's remark that the conic traits class has not been improved for a long time and thus, one might not consider it to be optimized code. Nevertheless, we believe that it is the internal use of the data type `CORE::Expr` (and thus, the usage of constructive separation bounds) which slows down the current CGAL traits class, despite possible optimizations.

The same is true for the comparison of rational functions. For fixed degree and bitsize, and an increasing number of curves, our approach is faster roughly by a constant factor, which becomes larger for higher degrees. If we fix the number of curves, we see that function of higher degree are handled much more efficiently. Again, we claim that the CGAL implementation suffers from the use of constructive separation bounds, which makes it become particularly slow for higher degrees.

[43] We thank Ron Wein for answering our questions regarding this traits class.

(n, τ, m)	#(V,E,F)	CGAL	CA	res	solve	lift	inter	comp.	decomp.	per vertex
(2,16,10)	(98, 220, 123)	0.18	**0.1**	3%	11%	14%	40%	0%	14%	1102
(2,16,40)	(1836, 3766, 1931)	3.18	**1.2**	11%	9%	8%	24%	17%	10%	657
(2,16,70)	(5066, 10298, 5233)	8.36	**3.44**	13%	8%	8%	23%	18%	13%	679
(2,16,100)	(8918, 18056, 9139)	13.67	**6.04**	16%	9%	6%	19%	19%	14%	677
(6,16,10)	(114, 254, 141)	1.56	**0.23**	20%	17%	12%	20%	8%	6%	2035
(6,16,40)	(2074, 4266, 2193)	27.11	**2.65**	21%	12%	7%	18%	19%	10%	1278
(6,16,70)	(7020, 14238, 7219)	94.78	**8.47**	19%	11%	4%	17%	23%	11%	1206
(6,16,100)	(13124, 26534, 13411)	169.95	**16.48**	20%	12%	5%	16%	25%	11%	1256
(3,16,20)	(422, 890, 469)	1.28	**0.43**	20%	17%	6%	20%	10%	14%	1033
(7,16,20)	(594, 1242, 649)	12.95	**0.81**	21%	9%	5%	18%	20%	12%	1367
(11,16,20)	(616, 1292, 677)	36.19	**1.45**	28%	15%	5%	19%	15%	7%	2357
(15,16,20)	(676, 1418, 743)	101.87	**2.05**	29%	14%	5%	18%	14%	8%	3041

Table 4.10. Benchmark results for m random rational functions of magnitude (n, τ).

After all, we take these comparisons as a proof of the maturation of our implementation (although further improvements are certainly possible), and as a proof of success for our principal approach of replacing symbolic computations by numerical ones as much as possible (but without using constructive separation bounds). Indeed, we yield an general and efficient algorithm for arrangement computation, that even outperforms existing implementations for restricted subclasses of algebraic curves.

We compared our implementation with other arrangement algorithms as well. In what follows, we report on our experience

`cad2d` As already mentioned, `cad2d` computes the *cad* for an arbitrary number of curves, thus, its result can be considered as an arrangement. However, the constructed *cad* is a much more complicated structure than the arrangement: at any critical x-coordinate, each input curve is stacked (i.e., its fiber is computed) and those fiber points are refined to disjointness (the output resembles a curve analysis of the union of all curves). In contrast, our algorithm always considers only two curves at once; for that reason, a comparison is not fair for a large number of input curves. In [EK08a], both approaches are compared for at most three input curves. The results affirm the results obtained for the one curve analysis, namely that `cad2d` has problems with singular curves, high coefficients, and also tangential intersections.

`CubiX` A library closely related to ours is `CubiX`, part of the EXACUS library [EKSW06], which was also designed at MPI. It computes arrangements of algebraic curves of degrees up to 3. The algorithmic framework is basically the same as that of our approach, unless that the methods for curve analysis and curve pair analysis are specialized variants for bounded degrees. In [EK08a], it was stated that `Cubix` and `AlciX` (a preliminary version of our implementation) showed roughly the same performance for cubic curves. We were not able to replicate those experiments for this thesis, because `CubiX` is no longer supported by our working group (partially because the general method presented here renders the cubic approach deprecated), and it was not possible to run `Cubix` on the platform where the benchmarks were performed.

Axel The Axel library[44] also offers an algorithm to compute arrangements. However, according to the authors, their method is currently only parameterized with an inexact solver and thus, does not return a certified answer.[45] Therefore, a comparison would only be of limited expressiveness, because we would be comparing an exact method with an approximate one. Moreover, we had problems running a stable version of Axel on our Linux workstation; the newest version is currently only available for Mac OS.

[44] `http://axel.inria.fr/`

[45] We thank Bernard Mourrain for this remark.

Es ist nicht genug, zu wissen, man muß auch anwenden; es ist nicht genug, zu wollen, man muß auch tun.
(In the end we retain from our studies only that which we practically apply.)
Johann Wolfgang von Goethe

5 Applications of Algebraic Arrangements

In this chapter, we consider several direct, and less direct generalizations regarding the arrangement algorithm (Chapter 3) and its software representation (Chapter 4).

In particular, we present a web application to visualize arrangements computed by our implementation (Section 5.1), we show how our software design can be extended to rotated algebraic curves (Section 5.2), and how it supports the computation of arrangements on orientable manifolds such as tori (Section 5.3). We sketch further work in progress, which is also based on the arrangement algorithm, in Section 5.4. Finally, another application, the exact topological analysis of algebraic surfaces, is treated in Chapter 6 separately.

5.1. A web application for visualizing algebraic arrangements

A natural second step after computing an arrangement in the plane is to provide an accurate picture of it, such that the user can explore its features visually. The goal is to bring together our algorithm with an exact visualization method for algebraic segments, and provide the result to a broader community via the World Wide Web. A video, together with an extended abstract presenting these results has appeared in [EK08c].

Emeliyanenko et al. [EBS09] describe an exact method for drawing x-monotone segments of algebraic plane curves. Given such a segment s, it begins with some so-called *seed point* in the interior of the segment. The segment is then traced in each direction towards its endpoints. The next pixel is determined by considering the box containing the current pixel and its 8 adjacent neighbors; in the best case, the curves intersects this box excalty twice, and the pixel where it leaves the box is taken as the next pixel. If the curve intersects the box boundary more than twice (because of other segments close-by, or because the segment reenters the box), local subdivision is performed to decrease the pixel size, eventually arriving at a definite answer. The method makes use of several interval arithmetic techniques , and adaptive precision methods. We refer to [EBS09] and [Eme07] for technical details. Once again, we underline that the method presented is a *local* drawing method that handles x-monotone segments separately. Therefore, it is also suitable to plot arrangements defined by x-monotone segments.

We run a public web server (`http://exacus.mpi-inf.mpg.de/cgi-bin/xalci.cgi`) to compute, visualize, and explore algebraic arrangements. The client sends the defining polynomials of the algebraic curves to our webserver. The server then computes the combinatorial arrangement description, and the image (in png format). Both are sent to the client who displays the result in the web browser using Flash.[46] With this architecture, the user is not burdened with installing additional software to use the application.

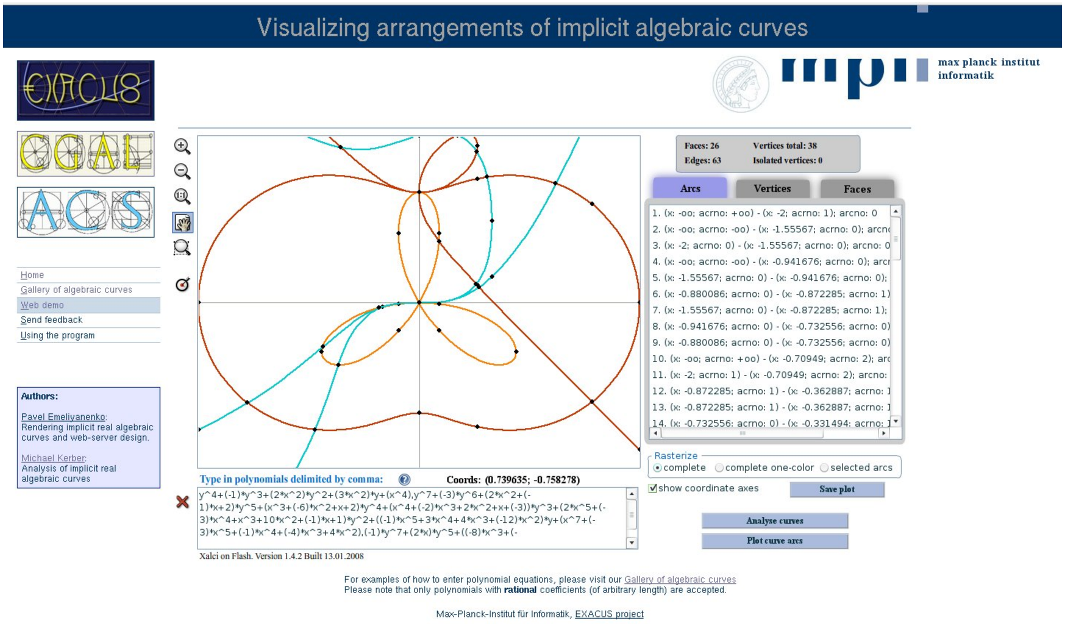


Figure 5.1. A screenshot of the webpage. In the bottom text field, the user types the defining polynomials. On the right, the arrangement information like the number of vertices, edges, and faces, and geometric information on every component is displayed. In the middle, the arrangement is visualized and can be explored via the buttons on the left of the plot.

We point out that we provide more output than "just" a reliable plot of the arrangement (which could possibly be rendered more quickly). Instead, the user has control over the arrangement structure and can explore the arrangement visually through the following interface.

 : zoom in by a factor of 2

 : zoom out by a factor of 2

 : reset to default zoom

 : focus on point – click the curve plot to center at a special point

 : focus on region – hold mouse button and select the area of interest

 : feature selection mode

[46] http://www.macromedia.com/software/flash/about/

Some of these features were used to produce Figure 5.2. However, a much more vivid introduction to the capabilities of the webpage is given in the video [EK08c], which is available from the homepage of the author.

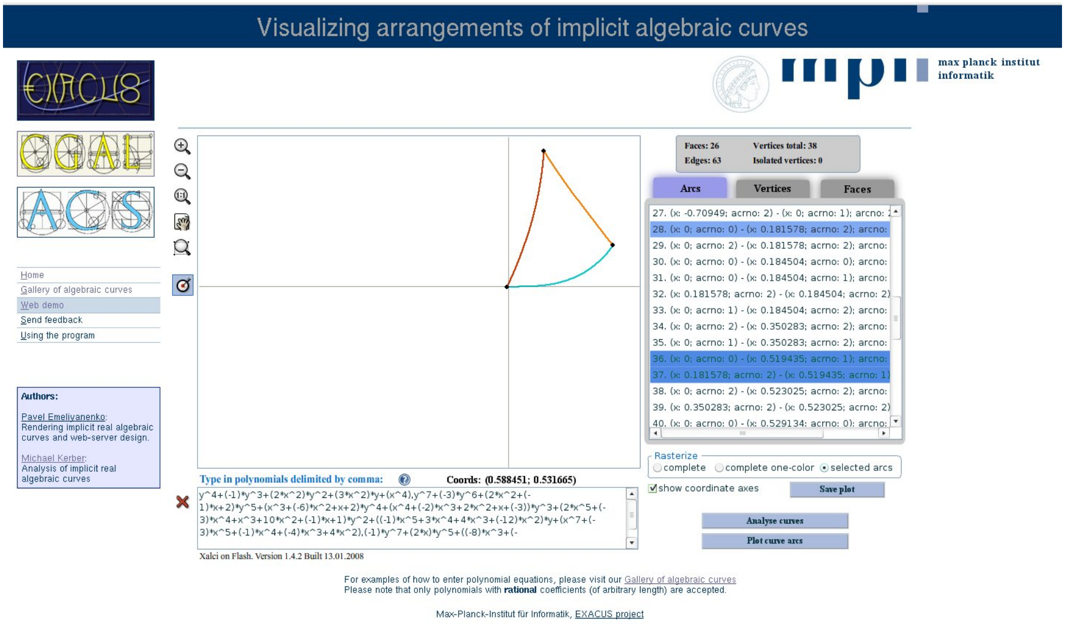


Figure 5.2. From Figure 5.1, we zoomed into some part of the arrangement. Also, we selected three arcs which are rasterized separately from the list on the right, and form a boundary cycle for a face of the arrangement.

Summary

The web application allows the user to use our algorithm directly, without a complicated initial installation process. Besides the fact that it is a good demonstration of our work, we believe that it might be a useful tool for educational purposes, or in situations where an unreliable plot might just not suffice. The gallery on the webpage shows several examples where the default plotting routine in MAPLE produces wrong pictures. In fact, the exact visualizer has become an indispensable debugging tool for the analysis of algebraic surfaces, as described in Section 6.

5.2. Arrangements of rotated curves

Our generic software design as presented on page 157 allows –in principle – instantiations of the algebraic kernels (uni- and bivariate) for types other than integer coefficients.[47] Nevertheless, the actual adaption to other number types is a non-trivial issue, since the

[47] We do not count curves with rational coefficients as a real generalization, since one can just clear denominators to get an integral curve.

underlying algebraic tools like square-free factorization, subresultant computations, and refinement of algebraic numbers have to be provided in a generic but still efficient form.

We next describe how the extension to more complicated domains allows the exact computation of arrangements of rotated algebraic curves. More precisely, curves defined by integer coefficients can be rotated by an angle that is constructible by compass and straightedge. This yields curves whoses defining polynomial has coefficients in a domain that includes nested square roots of integers. This work constitutes a generalization of [BCW07], who considered the same setup for conics.

Moreover, we have implemented a certified version of an approximation approach. This means that for any angle φ (in radian measure) and any precision ε, we compute the exact rotation of the curve by an angle φ' such that $|\varphi - \varphi'| < \varepsilon$. Thus "approximation" only means approximating the angle – the actual rotation is still performed in an exact manner and thus, the rotated curve is guaranteed to have the same topology as the original one. We obtain the approximated angle by deriving a certified version of the algorithm presented in [CDR92].

From a mathematical point of view, the rotation of an algebraic curve is not very complicated: let $V(f)$ be an algebraic curve and let φ denote the rotation angle; then the curve rotated counterclockwise by φ around the origin can be expressed as $V(f_{\text{rot}})$, where

$$f_{\text{rot}}(x, y) \quad = \quad f(\cos(\varphi)x + \sin(\varphi)y, -\sin(\varphi)x + \cos(\varphi)y). \tag{5.1}$$

In general, the coefficients of f_{rot} involve trigonometric functions and therefore, the resulting curve is not algebraic. Hence, we estimate that an exact and efficient algorithm dealing with general rotated curves is currently out of reach, if realizable at all. For that reason, we restrict the problem to certain well-behaved angles and to an approximation approach. We give some details about both methods in the following two subsections and also report on some benchmarks results we obtained when comparing the exact and the approximate approaches.

5.2.1. Rotation by exact angles

We exploit the fact that the sine and cosine of certain angles can be represented by square root expressions. This is possible if and only if the angle is constructible with compass and straightedge. The following well-known result characterizes all possibilities.

Theorem 5.2.1 (Gauss). *An angle α is constructible with compass and straightedge if and only if $\alpha = c \cdot \frac{360}{2^k p_1 \cdots p_s}$ where $c, k, s \in \mathbb{N}$ and $p_1, \ldots, p_s$ are distinct Fermat primes , that is, primes of the form $2^{2^m} + 1, m \geq 0$.*

We can use the following exact angles in our implementation.

- $\sin 45° = \frac{\sqrt{2}}{2}, \quad \cos 45° = \frac{\sqrt{2}}{2} \in \mathbb{Q}[\sqrt{2}]$
- $\sin 30° = \frac{1}{2}, \quad \cos 30° = \frac{\sqrt{3}}{2} \in \mathbb{Q}[\sqrt{3}]$
- $\sin 18° = \frac{1}{4}\left(\sqrt{5} - 1\right), \quad \cos 18° = \frac{1}{4}\sqrt{10 + 2\sqrt{5}} \in \mathbb{Q}[\sqrt{5}, \sqrt{10 + 2\sqrt{5}}]$
- $\sin 15° = \frac{1}{4}\left(\sqrt{6} - \sqrt{2}\right), \quad \cos 15° = \frac{1}{4}\left(\sqrt{2} + \sqrt{6}\right) \in \mathbb{Q}[\sqrt{2}, \sqrt{3}]$
- $\sin 9° = \frac{1}{16}\left((\sqrt{10} - \sqrt{2})\sqrt{10 + 2\sqrt{5}} + 2\sqrt{10} + 2\sqrt{2}\right)$
  $\cos 9° = \frac{1}{16}\left((\sqrt{2} - \sqrt{10})\sqrt{10 + 2\sqrt{5}} + 2\sqrt{10} + 2\sqrt{2}\right) \in \mathbb{Q}[\sqrt{2}, \sqrt{5}, \sqrt{10 + 2\sqrt{5}}]$

- $\sin 6° = \frac{1}{16}\left((\sqrt{3}-\sqrt{15})\sqrt{10+2\sqrt{5}}+2+2\sqrt{5}\right)$
  $\cos 6° = \frac{1}{16}\left((\sqrt{5}-1)\sqrt{10+2\sqrt{5}}+2\sqrt{3}+2\sqrt{15}\right) \in \mathbb{Q}[\sqrt{3},\sqrt{5},\sqrt{10+2\sqrt{5}}]$
- $\sin 3° = \frac{1}{16}\left(\sqrt{2}-\sqrt{6}-\sqrt{10}+\sqrt{30}+(\sqrt{2}+\sqrt{6})\sqrt{10+2\sqrt{5}}\right)$
  $\cos 3° = \frac{1}{16}\left((\sqrt{2}+\sqrt{6})\sqrt{10+2\sqrt{5}}+\sqrt{2}-\sqrt{6}-\sqrt{10}+\sqrt{30}\right)$
  $\in \mathbb{Q}[\sqrt{2},\sqrt{3},\sqrt{5},\sqrt{10+2\sqrt{5}}]$

Note how the domain for the sine and cosine becomes more complicated for smaller angles. By the addition theorems

$$\begin{aligned}\sin(\alpha+\beta) &= \cos\alpha\sin\beta+\sin\alpha\cos\beta\\ \cos(\alpha+\beta) &= \cos\alpha\cos\beta-\sin\alpha\sin\beta,\end{aligned}$$

and it follows that the sine and cosine of multiples of an angle remain in the same domain as well, and that they are easily computable.

We now enable algebraic kernels for all coefficient domains defined in the above enumeration. For that, we make use of CGAL's `Sqrt_extension` class (contained in the package `Number_types` [HHK⁺08]). It is a template class with two parameters: `Sqrt_extension<`A`,`B`>` represents all numbers of type $a_1 + a_2 \cdot \sqrt{b}$, where $a_1, a_2 \in A$ and $b \in B$. A problem lies in the interoperability of such numbers; for instance, $\sqrt{2}$ and $\sqrt{3}$ are both representable by `Sqrt_extension<I,I>` (where `I` is a type representing $\mathbb{Z}$), but $\sqrt{2}+\sqrt{3}$ is no longer of this type. Avoiding such unpleasant behavior requires us to fix the root expression for each `Sqrt_extension` during the computation; in other words, `Sqrt_extension<I,I>` can only be used to model $\mathbb{Z}[\sqrt{a}]$ for a fixed $a \in \mathbb{Z}$.

However, the type can be cascaded to model more demanding domains. For instance, to realize the domain $\mathbb{Z}[\sqrt{2},\sqrt{3}]$, as in the example above, one can use the type

```
Sqrt_extension<Sqrt_extension<I,I>,I>.
```

This represents numbers of type $c_1 + c_2\sqrt{b_1}$, where c_1 and c_2 are again square root expressions of type $a_1 + a_2\sqrt{b_2}$. Note that there is an outer root b_1 and an inner root b_2 for the type; if all instances get consistent outer and inner roots (for instance $b_1 = 2$, $b_2 = 3$, or vice versa), then the domain $\mathbb{Z}[\sqrt{2},\sqrt{3}]$ can be modeled in this way.

It is worth mentioning that algebraic computations like (modular) gcd or subresultant computations have been provided for arbitrarily cascaded `Sqrt_extension` types. For gcd computations, this constituted a non-trivial research topic; see the work by Hemmer and Hülse [HH09]. This approach "only" computes the gcd up to a constant factor, and so, derived objects like the content of a bivariate polynomial, or the square-free part of a univariate polynomial are also only known up to a constant. This does not pose any theoretical difficulties in our algebraic kernel (since we are only interested in the roots of such polynomials), but it underlines that some efforts were required to extend the software to more general domains.

Let us go back to the original problem of computing the rotation of a curve. Fix a *base angle* $\varphi \in \{3°, 6°, 9°, 15°, 18°, 30°, 45°\}$. Choose $\mathbb{D}$ such that $\sin\varphi$ and $\cos\varphi \in \mathbb{D}$, and model $\mathbb{D}$ by a proper instantiation of `Sqrt_extension`; for instance, $\mathbb{D} = \mathbb{Q}[\sqrt{5},\sqrt{10+2\sqrt{5}}]$ (for angle 18°) is modelled by

```
Sqrt_extension< Sqrt_extension<Q,I>, Sqrt_extension<I,I> >
```

where `Q` models rational numbers, and `I` models integers. In the same way, we can model $\mathbb{D}_{\mathbb{Z}} = \mathbb{Z}[\sqrt{5}, \sqrt{10+2\sqrt{5}}]$, by replacing `Q` with `I`. For a multiple α of φ and $f \in \mathbb{Z}[x,y]$, the rotated polynomial f_{rot} is an element of $\mathbb{D}_{\mathbb{Z}}[x,y]$. f_{rot} is computed by looking up $\sin\alpha$ and $\cos\alpha$ in a hard-coded list, substituting $(\cos(\alpha)x + \sin(\alpha)y)$ for x and $(-\sin(\alpha)x + \cos(\alpha)y)$ for y, and finally clearing the denominators.

Next, we introduce the new template class `Rotated_algebraic_kernel_2` for rotated curves. Its main functionality is based on the class template `Algebraic_kernel_2` instantiated with a suitable coefficient type. A single instantiation is dedicated to a fixed base angle, given by a template parameter. We provide specializations for every integer angle constructible with compass and straightedge, that is, 45°, 30°, 18°, 15°, 9°, 6°, and 3°. For convenience, we added some types and methods: Recall that the `Algebraic_kernel_2` contains a functor `Construct_curve_2` to construct a curve analysis object out of a defining equation. `Rotated_algebraic_kernel_2` extends this functor by a new construction method for rotated curves:

```
struct Construct_curve_analysis_2 {
  ...
  Curve_analysis_2 operator()
    (const Unrotated_polynomial_2& f, const Angle& angle) const;
};
```

The functor rotates the curve `f` by `angle` and analyzes the rotated curve (the two types `Unrotated_polynomial_2` and `Angle` are introduced in the rotated kernel, with the obvious meaning).

The ability of our generic algebraic kernel to handle square root numbers can serve applications other than rotating curves. For instance, our kernel enables the computation of arrangements on surfaces whose parameterizations involve algebraic coefficients [DHPS07]. Also, our method to stratify and triangulate an algebraic surface as presented in Chapter 6 comes into reach for surfaces with square root coefficients (a prominent example for this is Boy's surface[48]).

5.2.2. Rotations by approximate angles

We begin with the exact definition of the problem: Given some angle φ and some precision $p \in \mathbb{Z}$, find rational values s' and c' such that $s' = \sin\varphi'$ and $c' = \cos\varphi'$ for some φ' with $|\varphi' - \varphi| < 2^p$. In addition, the values s' and c' should be as small as possible (in terms of their bitsize), because their bitsize affects the bitsize of f_{rot} as defined in (5.1).

We recall the approach of Canny et al. [CDR92] to solve this problem. Their algorithm relies on Pythagorean triples and approximations by continued fractions. Here is their solution: Write $c' = \frac{a}{d}$ and $s' = \frac{b}{d}$ with common denominator d. Since $c'^2 + s'^2 = 1$, a, b, and d form a *Pythagorean triple*. Such triples are generated as: $a = n^2 - m^2$, $b = 2nm$ and $c = n^2 + m^2$, where $n, m \in \mathbb{N}$. Thus, by choosing $t' := \frac{n}{m}$ arbitrarily, we obtain

$$s' = \frac{2mn}{n^2+m^2} = \frac{2}{t' + \frac{1}{t'}} \text{ and } c' = \frac{n^2-m^2}{n^2+m^2} = \frac{t' - \frac{1}{t'}}{t' + \frac{1}{t'}}.$$

[48] `http://mathworld.wolfram.com/BoySurface.html`

How to choose t' such that s' and c' are good approximations of $\sin\varphi$ and $\cos\varphi$? For that, consider the value $t \in \mathbb{R}$ that precisely yields $\sin\varphi$ and $\cos\varphi$, which is

$$t = \frac{1}{\sin\varphi} + \sqrt{\frac{1}{\sin^2\varphi} - 1}.$$

In general, this is an irrational number. However, the idea is to approximate t by a sequence of rational values until the arcussine of s' has a distance of at most 2^p from φ. For that approximation, a *continued fraction expansion* of t is computed by a Euclidean-like scheme, which produces much smaller results compared to a bisection approach. Canny et al. also propose some optimizations to get even slightly smaller values of c' and s', but they report only marginal improvements. Thus, we only consider the approach summarized in Algorithm 5.1.

Algorithm 5.1. Approximate rotation, non-certified version

procedure ROTATE_APPROX(φ, p)
    $s \leftarrow \sin(\frac{\varphi\pi}{180})$ ▷ Converted angle to radian measure first
    $t \leftarrow \frac{1}{s} + \sqrt{\frac{1}{s^2} - 1}$
    $e_0 \leftarrow t, p_0 \leftarrow 0, q_0 \leftarrow 1, e_1 \leftarrow -1, p_1 \leftarrow 1, q_1 \leftarrow 0$
    repeat
        $r \leftarrow \lfloor \frac{e_0}{e_1} \rfloor$
        $e_0^* \leftarrow e_0, p_0^* \leftarrow p_0, q_0^* \leftarrow q_0$ ▷ temporarily stored
        $e_0 \leftarrow e_1, p_0 \leftarrow p_1, q_0 \leftarrow q_1$
        $e_1 \leftarrow e_0^* - re_1, p_1 \leftarrow p_0^* - rp_1, q_1 \leftarrow q_0^* - rq_1$
        $t' \leftarrow \frac{p1}{q1}$ ▷ t' approximates t
        $s' \leftarrow \frac{2}{t' + \frac{1}{t'}}$ ▷ s' approximates s
    until $|\arcsin(s')\frac{180}{\pi} - \varphi| < 2^p$ ▷ Convert angle into degree measure
    $c' \leftarrow \frac{t' - \frac{1}{t'}}{t' + \frac{1}{t'}}$
    return (s', c')
end procedure

When Algorithm 5.1 is executed with double precision (or with any other fixed precision), it is not guaranteed that the computed rotation angle is indeed at a distance of at most 2^p from the input angle, due to possible rounding errors. We use an adaptive precision technique and interval arithmetic to certify the termination condition of the loop in Algorithm 5.1. The value of t is computed approximately, before the loop starts. We do not bound the distance from the computed value $\tilde{t}$ to the exact t, but it is guaranteed that this distance converges to zero as the precision increases. In the loop, the value of t' approximates $\tilde{t}$, instead of t itself. The arcussine computation and the check for the termination condition is performed with interval arithmetic. However, we need a second termination condition in case the current precision is not sufficient to find a "good" rotation. If so, the rational t' will finally be equal to $\tilde{t}$ since the continued fraction sequence of a rational value is finite and $\tilde{t}$ is rational. In this case, we increase the precision, recompute $\tilde{t}$, and restart the loop.

Algorithm 5.2 summarizes our certified approach. The suboperations π, sin, arcsin, and sqrt are performed using interval arithmetic. They all receive a second parameter p

and return an interval with width less than 2^p containing the exact value of the operation. How to obtain such intervals is explained in Appendix A. In addition, we use a method $\chi(I)$ that chooses some element of an interval (e.g., the midpoint).

Algorithm 5.2. Approximate rotation, certified version

1: **procedure** ROTATE_APPROX_CERTIFIED(φ, p)
2: $\quad prec \leftarrow 16$ $\qquad \triangleright$ working precision
3: $\quad pi \leftarrow \pi(prec)$ $\qquad \triangleright$ pi approximates π
4: $\quad \tilde{s} \leftarrow \chi(\sin(\frac{\varphi \cdot \chi(pi)}{180}, prec))$
5: $\quad \tilde{t} \leftarrow \frac{1}{\tilde{s}} + \chi(\mathrm{sqrt}(\frac{1}{\tilde{s}^2} - 1, prec))$
6: $\quad e_0 \leftarrow \tilde{t}, p_0 \leftarrow 0, q_0 \leftarrow 1, e_1 \leftarrow -1, p_1 \leftarrow 1, q_1 \leftarrow 0$
7: $\quad$ **repeat**
8: $\qquad r \leftarrow \lfloor \frac{e_0}{e_1} \rfloor$
9: $\qquad e_0^* \leftarrow e_0, p_0^* \leftarrow p_0, q_0^* \leftarrow q_0$
10: $\qquad e_0 \leftarrow e_1, p_0 \leftarrow p_1, q_0 \leftarrow q_1$
11: $\qquad e_1 \leftarrow e_0^* - re_1, p_1 \leftarrow p_0^* - rp_1, q_1 \leftarrow q_0^* - rq_1$
12: $\qquad t' \leftarrow \frac{p1}{q1}$ $\qquad \triangleright$ t' approximates $\tilde{t}$
13: $\qquad$ **if** $t' = \tilde{t}$ **then** $prec \leftarrow 2prec$; **goto** 3; **end if**
14: $\qquad s' \leftarrow \frac{2}{t+\frac{1}{t}}$ $\qquad \triangleright$ s' approximates $\tilde{s}$
15: $\quad$ **until** $|\arcsin(s', prec)\frac{180}{pi} - \varphi| < 2^p$ $\qquad \triangleright$ Interval arithmetic
16: $\quad c' \leftarrow \frac{t'-\frac{1}{t'}}{t'+\frac{1}{t'}}$
17: $\quad$ **return** (s', c')
18: **end procedure**

Our actual implementation differs in two ways from the description of Algorithm 5.2. First, if the interval $(\varphi - 2^{p-1}, \varphi + 2^{p-1})$ contains a multiple of 90°, sine and cosine are immediately set to 0 and ± 1 (depending on the angle). Also, steps 2-15 are not performed for φ itself but for a corresponding angle φ' between 0° and 45°. The sine and cosine of φ can be reduced to the sine and cosine of φ'. To give an example, $\sin(289°) = -\cos(19°)$ and $\cos(289°) = \sin(19°)$, so the sine and cosine of 289° can be reduced to the sine and cosine of 19°. The underlying trigonometric functions have been implemented internally. By the optimization just mentioned, all trigonometric functions are called with an argument between 0 and $\frac{\pi}{4}$. This allows a slightly improved computation. Details are explained in Appendix A. It is not within the scope of this work to implement them in the best possible way — the efficient computation of trigonometric functions is a research topic in its own. Mainly, our goal here is to have a generic and certified implementation for these functions. We are not aware of an existing state-of-the-art implementation that combines efficiency with these two properties.

To construct curves rotated by approximate angles, we provided an additional functor in the `Algebraic_kernel_2`. It receives an unrotated polynomial f, a rotation angle φ, and an approximation precision p as parameters, and constructs the curve induced by f after (exactly) rotating it by an angle φ', such that $|\varphi' - \varphi| < 2^p$.

Compare the pros and cons of the approximation approach, with those of the exact approach in Section 5.2.1. Because the coefficients of the rotated curve remain rationals (or integers, after clearing denominators), arbitrary rotations are possible, instead of being fixed at predeterminate multiples of a base angle as in the exact approach. However, the price to pay is that the angle (not the rotation) is approximated. This might lead to a wrong topology in an arrangement of rotated curves, for instance, if two (exactly) rotated curves intersected tangentially.

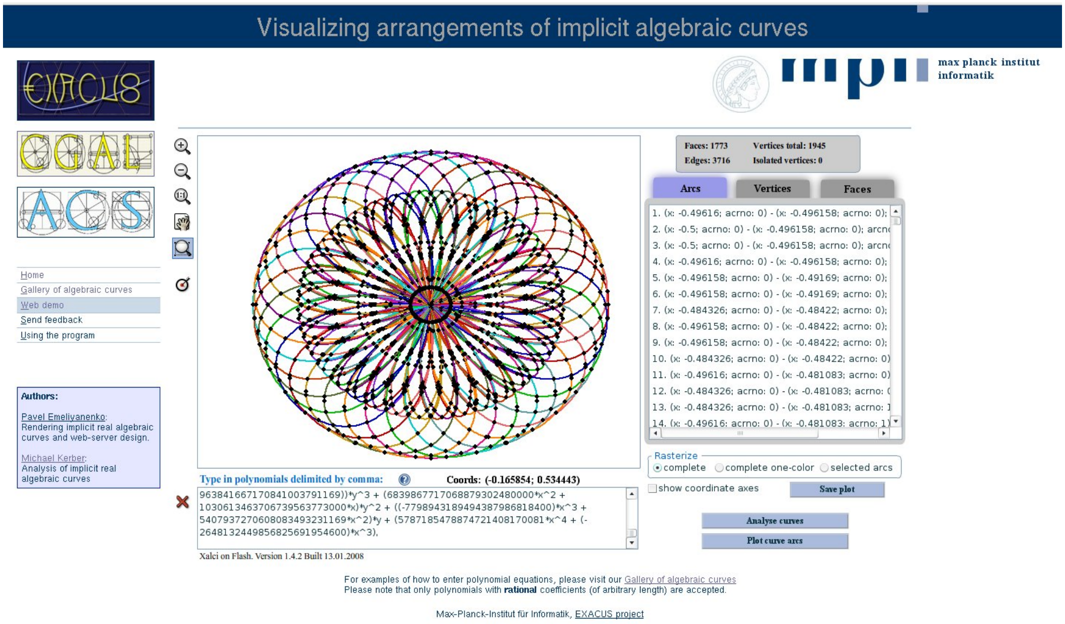


Figure 5.3. The arrangement of 36 rotations of the curve $V(2x^4 + y^4 - x^3 + xy^2)$, each rotation being by a multiple of $10°$. The approximate approach is used to produce the picture (the visualizing algorithm described in Section 5.1 is currently limited to integer coefficients).

We remark that the `Kernel_23` package of CGAL contains a function with the name `rational_rotation_approximation` that is also based on the approach [BFG+08]. That method differs from ours in that that it gives approximation guarantees for the computed sine value instead of the rotation angle.

5.2.3. Experimental results

We have experimentally compared both the exact and the approximate approach on the same machine and the same setup as in Section 4.3. Table 5.1 shows the time spent for a curve analysis of rotated curves for random curves of type rand(n,c) (as defined in Section 4.3). Each curve was rotated by $30°$. In order to show the overhead due to more complicated coefficient types, the computation was performed with base angles of $30°, 15°, 6°$ and $3°$. We compared the results with the approximate approach. Although Algorithm 5.2 cannot be considered optimized since it currently uses rather naive algorithms to approximate trigonometric functions, the time needed to compute the defining polynomial of a rotated curve was negligible in all instances tested.

One can observe that smaller base angles lead to a significant performance loss in the

	Exact				Approximate			
Curve	30°	15°	6°	3°	10 bits	40 bits	70 bits	100 bits
rand(6,50)	0.12	0.2	0.3	0.56	0.04	0.11	0.6	0.82
rand(9,50)	0.7	1.21	1.52	2.97	0.33	0.86	2.09	3.15
rand(12,50)	2.78	4.64	5.34	10.19	1.95	5.97	12.22	19.54
rand(15,50)	9.33	14.74	15.64	28.2	8.71	28.02	57.17	91.38
rand(18,50)	28.66	40.52	41.02	66.82	30.53	99.82	202.2	325.2
rand(10,64)	1.16	1.85	2.29	4.29	0.64	1.78	3.87	5.99
rand(10,128)	2.04	2.85	3.32	5.58	0.88	2.12	4.26	6.43
rand(10,256)	4.26	4.97	5.39	7.62	1.39	2.8	5.05	7.42
rand(10,512)	9.57	10.69	11.12	13.38	2.62	4.33	6.92	9.46
rand(10,1024)	28.5	29.65	30.06	32.9	5.92	8.07	11.12	14.28

Table 5.1. Benchmark results for random curves rotated by 30°. On the left, we used the exact approach using different base angles. On the right, we used the approximate approach using different approximation qualities.

exact method due to working with a more complicated coefficient type. On the other hand, the approximate approach clearly suffers when improving the quality of the approximations. Moreover, the exact approach has a significant advantage for high degree curves. The reason is that for τ being the bitsize of the sine and cosine approximations, the coefficient size of the rotated curve is increased by $O(n\tau)$, where n is the degree of the polynomial. Of course, this has a considerable effect in the subsequent algebraic computations. In particular, the bitsize of the resultant increases by $\tilde{O}(n^2\tau)$.

In contrast, the approximate approach is less vulnerable to high coefficient sizes in the input polynomials, as the second part of Table 5.1 shows.

	degree 6				degree 12			
	45°	30°	18°	15°	45°	30°	18°	15°
exact	1.66	2.15	19.88	7.02	53.19	61.11	1385.05	267.17
approx. prec. 10	0.69	0.71	0.68	0.68	39.53	38.15	37.62	35.89
approx. prec. 40	1.23	1.27	1.25	1.28	114.29	115.32	114.48	116.78
approx. prec. 70	2.10	2.10	2.26	2.23	216.18	219.07	237.64	227.41
approx. prec. 100	3.12	3.19	3.18	3.07	359.80	356.49	365.64	349.07

Table 5.2. Running times (in seconds) for rotated arrangements of 5 curves of type rand(10,50).

We also computed the arrangement of rotated curves. We consider five fixed random curves of type rand(10,50). Table 5.2 shows the time to compute an arrangement of these curves, rotated by a given angle. In this case, we did not consider the time for computing the approximate angle in the timings. The performance of the approximation approach is roughly the same for all angles – not surprisingly, since all those arrangements are expected to be equally difficult. The rotation by 18° shows the worst performance with the exact approach. We think that this is due to the complicated coefficient type involving nested square root expressions used in this case. Compared to the approximate approach, we observe that for angles 45° and 30°, the exact computation is competitive even for rather coarse approximations, and it becomes more competitive at higher degrees.

Summary

Owing to the careful generic design of our algebraic kernel, the analysis of rotated curves and computing their arrangement is possible with the same code as that for unrotated curves. Exact rotations are possible if the rotation angles are confined to multiples of a common base angle. A performance penalty is measurable for small base angles, which is no surprise because of the more complicated computation domain. Still, we have demonstrated that rahter complicated arrangements are computable and that the exact approach is sometimes even competitive with the approximation approach, which rotates by an approximate angle.

5.3. Arrangements on tori and Dupin cyclides

Let S be a surface, with or without boundary, and consider a set of curves $c_1, \ldots, c_m$ on S. These curves decompose the surface into connected regions; thus, the arrangements on S induced by $c_1, \ldots, c_m$ can be defined in just the same way as in the plane. On orientable manifolds,[49] such arrangements are also representable via a DCEL structure. Recently, the `Arrangement_2` package of CGAL has been extended to the `Arrangement_on_surface_2` package that allows one to construct and maintain arrangements on parameterized surfaces embedded in $\mathbb{R}^3$ [BFH$^+$07] [BFH$^+$09b]. Applications of this package are discussed in [BFH$^+$09a]: a distinguished usecase are arrangements of great circles on a sphere. They lead to applications like the Minkowski sum of convex polyhedra, the (upper or lower) envelope of surfaces, and Voronoi diagrams on spheres. We also refer to [FSH08] for a video presentation of these applications.

Beyond great circles on spheres, the framework can also be used for more general curves on more complicated surfaces. Consider the following setup: Let S be a fixed *reference surface* and let $\{S_1, \ldots, S_n\}$ be other algebraic surfaces (of arbitrary degree) not overlapping with S. The intersection $S \cap S_i$ then defines a curve on S, and the set of all such intersection curves induces an arrangement on S.

This problem has been considered for various choices of reference surfaces: The case of *elliptic quadrics*, that is, ellipsoids, elliptic paraboloids, and elliptic cylinders (Figure 5.4) has been considered by Berberich et al. [BFH$^+$07]. Their solution exploits the relative simplicity of the underlying reference surface (mainly, that such surfaces are quadrics which have an upper part and a lower part).

We will consider another reference surface, namely the case of a *ring Dupin cyclide*, which is the generalization of a torus [BK08] [BFH$^+$09a] [Ber08, §4.6]. To compute arrangements on such a cyclide, we represent the intersection curves as algebraic curves in the parameter space $\mathbb{R}^2$ and compute their arrangement using the method for planar curves. Curves with high degrees arise naturally with this approach. For instance, the projection of the intersection curve of two tori already is of total degree 16.

In principle, the approach that we present is not restricted to Dupin cyclides but can be used for any reference surface that allows a rational parameterization (in particular,

[49] Intuitively, a surface is orientable if it has two distinct sides. The most famous example of a non-orientable surface is a Möbius strip.

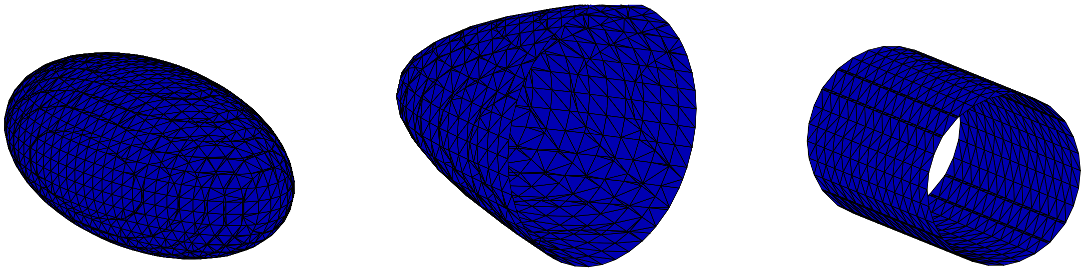


Figure 5.4. Ellipsoid, elliptic paraboloid, and elliptic cylinder (left to right)

the elliptic quadric case could also be solved by this approach). One "just" has to take into account the special topology of the reference surface when constructing the DCEL structure for the arrangement. This is a non-trivial issue for the Dupin cyclide, as we will see in Section 5.3.3. Also, we remark that the degrees of the algebraic curves in the parameter space constitute the limit of the practical usability of the approach.

We continue as follows in the rest of this section: The CGAL package to compute arrangements on surfaces is presented in Section 5.3.1. Some facts about Dupin cyclides are collected in Section 5.3.2. Then, we will describe our approach in Section 5.3.3 and report on some experimental results in Section 5.3.4.

5.3.1. The `Arrangements_on_surface_2` framework

CGAL's `Arrangements_on_surface_2` package provides an arrangement class that can be used to construct, maintain, overlay, and query two-dimensional arrangements on a parametric surface. By "parametric surface", we mean a surface S that can be parameterized by a rectangular parameter space, which means that there exists a surjective continuous mapping

$$P_S : [u_0, u_1] \times [v_0, v_1] \to S,$$

with u_0, u_1, v_0, v_1 as either real values or $\pm\infty$. Moreover, P_S must be one-to-one in the interior of the parameter space. The boundary consists of a rectangle; each side is called a *boundary side*. At each boundary side, the following special features can occur:

1. **Contraction:** All points of the boundary side are mapped to the same point of S.
2. **Identification:** Each point of the boundary side is mapped to the same point as the corresponding point on the opposite boundary side.

The contractions and identifications on the boundaries determine the topological type of the surface. For instance, a sphere (and an ellipsoid) have contractions at the top and bottom boundaries, and an identification at the left and right boundaries. A cylinder only has an identification at the left and right boundaries, and a torus and a Dupin cyclide have identifications on the left and right boundaries as well as on the top and bottom boundaries.[50]

To compute arrangements, the framework conceptually performs a sweep in the parameter space, that is, a line $u = u_0$ is swept to the right through the parameter space. The actual sweep-"line" is the image of $u = u_0$ on the surface S under P_S. The correct

[50] The framework does not support inverse identifications, appearing, for instance, in a Möbius strip.

intuition for a sphere is to sweep with a meridian that rotates around the sphere. For a torus, the intuition is to sweep with a circle along the tube of the torus.

While sweeping, the DCEL structure is built up, similar to the case of planar arrangements. However, special diligence is needed for several reasons. First of all, only one DCEL vertex must be created for all points at a contracted boundary side, and only one DCEL vertex must be created for each pair of points at identified boundary sides. Moreover, detecting boundary cycles of DCEL faces becomes more complicated than in the plane. For instance, the *Jordan curve theorem* does not hold on a torus, which means that a closed cycle does not necessarily split one face into two faces. One has to cope with such peculiarities for the given topological type of the reference surface.

The `Arrangement_on_surface_2` package tackles these problems by the following modular framework. Models of two concepts must be provided as template parameters.

`GeometryTraits`: A proper instantiation for this parameter has to be a model of CGAL's `ArrangementTraits_2` concept (page 157). To repeat its main characteristics, it defines the types `Curve_2`, `X_monotone_curve_2`, and `Point_2` and also some operations on them: Curves are split into x-monotone subcurves, points can be compared lexicographically, and the intersections of x-monotone curves are computed (as described on page 97).

`TopologyTraits`: An instantiation of this class is responsible for construct a DCEL structure that is consistent with respect to the topology of the reference surface. In particular, this means correctly creating and maintaining DCEL vertices at the boundary, according to identifications and contractions, and also keeping a consistent face structure whenever a loop is closed. For further reading about the internals of the package, we refer the reader to [BFH$^+$07] [BFH$^+$09b].

5.3.2. Dupin cyclides

Dupin introduced *cyclides* as surfaces whose lines of curvature are all circles [Dup22]. Later, the term "cyclide" was used for quartic surfaces with the circle at infinity as double curve [For12]; Dupin's cyclides have been called *Dupin cyclides* instead. In this work, we only consider Dupin cyclides and use the term *cyclide* according to Dupin's definition for shorter notation. Dupin cyclides are the generalization of the "natural" geometric surfaces like planes, cylinders, cones, spheres, and tori, what makes them useful for applications in solid modeling; compare [CDH89] [Pra90] [Boe90] [Joh93] [Pra95]. Most of the material in this section appears more detailed in [Büh95, § 1].

Maybe the most intuitive way of constructing a (Dupin) cyclide goes back to Maxwell; we cite it from Boehm [Boe90]:

> Let a sufficiently long string be fastened at one end to one focus f of an ellipse, let the string be kept always tight while sliding smoothly over the ellipse, then the other end z sweeps out the whole surface of a cyclide Z.

Note that choosing a circle in this construction yields a torus. We will assume that the cyclide is in a *standard position and orientation*, that is, the chosen base ellipse is defined by

$$(x/a)^2 + (y/b)^2 = 1, \quad a \geq b > 0.$$

The cyclide is defined uniquely by a, b, and a parameter μ that is the length of the string minus a. However, the cyclide can have self-intersections. We define $c = \sqrt{a^2 - b^2}$, which is the distance between the focus and the center of the ellipse. If $c < \mu \leq a$, we get a *ring cyclide* which is a surface without self-intersections. Otherwise, we get either a so-called *horned cyclide* (for $0 < \mu \leq c$), or a *spindle cyclide* (for $\mu > a$); cf. [Bez07]. We can only handle ring cyclides in our algorithm, so we always assume that $c < \mu \leq a$ is satisfied. Figure 5.5 shows two examples.

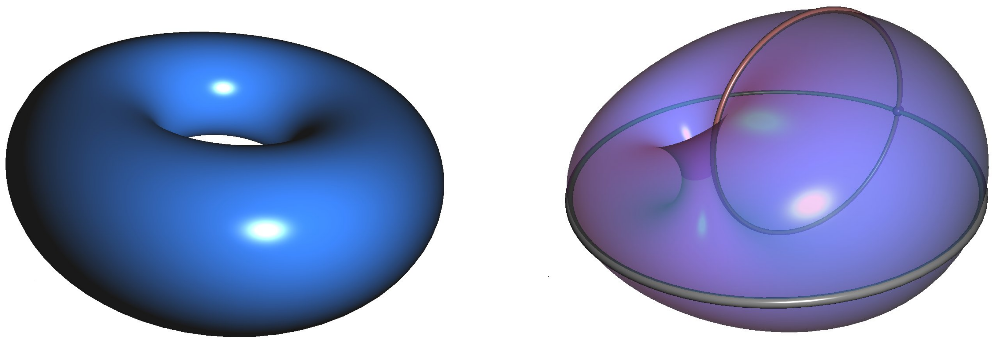


Figure 5.5. On the left: cyclide with $a = 2$, $b = 2$, $\mu = 1$. On the right: cyclide with a=13, $b = 12$, $\mu = 11$ and its cut circles.

A parameterization of the cyclide goes back to Forsyth [For12]. He also gave the following two alternatives for an implicit equation of the cyclide:

$$\begin{aligned} (x^2 + y^2 + z^2 - \mu^2 + b^2)^2 &= 4(ax - c\mu)^2 + 4b^2y^2 \\ (x^2 + y^2 + z^2 - \mu^2 - b^2)^2 &= 4(cx - a\mu)^2 - 4b^2z^2. \end{aligned}$$

With these equations, it is easy to prove [Joh93] that the intersection of the cyclide with the plane $y = 0$ consists of the two circles:

$$(x + a)^2 + z^2 = (\mu + c)^2 \tag{5.2}$$
$$(x - a)^2 + z^2 = (\mu - c)^2, \tag{5.3}$$

and the intersection with $z = 0$ are the two circles

$$(x + c)^2 + y^2 = (a + \mu)^2 \tag{5.4}$$
$$(x - c)^2 + y^2 = (a - \mu)^2. \tag{5.5}$$

In the case of a ring cyclide, the interiors of (5.2) and (5.3) are always disjoint, and the circle (5.5) is contained in the interior of (5.4).

The parameterization of the cyclide is given by

$$\begin{pmatrix} \varphi \\ \psi \end{pmatrix} \mapsto \begin{pmatrix} \frac{\mu(c - a\cos\varphi\cos\psi) + b^2\cos\varphi}{a - c\cos\varphi\cos\psi} \\ \frac{b(a - \mu\cos\psi)\sin\varphi}{a - c\cos\varphi\cos\psi} \\ \frac{b(c\cos\varphi - \mu)\sin\psi}{a - c\cos\varphi\cos\psi} \end{pmatrix}$$

with $\varphi, \psi \in [-\pi, \pi]$. We investigate which portion of the cyclide is parameterized at the boundaries of the parameter space:

Lemma 5.3.1. *If $\varphi = \pi$ or ($\varphi = -\pi$) is fixed, the parameterization above yields the circle $(x+a)^2 + z^2 = (\mu + c)^2$. If $\psi = \pi$ (or $\psi = -\pi$) is fixed, it yields the circle $(x+c)^2 + y^2 = (a+\mu)^2$. We call these circles the tube circle and the outer circle, respectively.*

Proof. Fix $\varphi = \pi$. This yields the parameterization

$$\psi \mapsto \begin{pmatrix} \frac{\mu(c+a\cos\psi)-b^2}{a+c\cos\psi} \\ 0 \\ \frac{-b(c+\mu)\sin\psi}{a+ccos\psi} \end{pmatrix}$$

Since the denominator does not vanish, this parameterizes a closed path in the plane $y = 0$, so it must be one of the circles (5.2) or (5.3) By setting $\psi = \pi$, we get the point $(-\mu - c - a, 0, 0)$, so it must be circle (5.2). The same argument can be used for $\psi = \pi$. $\square$

The tube circle and the outer circle meet at the point $p := (-\mu - c - a, 0, 0)$. We call this point the *pole* of the cyclide. Our application needs a rational parameterization of the cyclide without trigonometric functions. We use the standard trick to get rid of these functions (cf. [Gal01]): Using the identities

$$\cos\theta = \frac{1-\tan^2\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}} \quad \sin\theta = \frac{2\tan\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}},$$

we set $u := \tan\frac{\varphi}{2}, v := \tan\frac{\psi}{2}$. This yields

$$P : \mathbb{R}^2 \to \mathbb{R}^3, \begin{pmatrix} u \\ v \end{pmatrix} \mapsto \begin{pmatrix} \frac{\mu(c(1+u^2)(1+v^2)-a(1-v^2)(1-u^2))+b^2(1-u^2)(1+v^2)}{a(1+u^2)(1+v^2)-c(1-u^2)(1-v^2)} \\ \frac{2u(a(1+v^2)-\mu(1-v^2))b}{a(1+u^2)(1+v^2)-c(1-u^2)(1-v^2)} \\ \frac{2v(c(1-u^2)-\mu(1+u^2))b}{a(1+u^2)(1+v^2)-c(1-u^2)(1-v^2)} \end{pmatrix}.$$

The image of P is the cyclide without the tube circle and the outer circle. By setting $\varphi = \pi$ (or $\psi = \pi$) and applying the same trick, we also obtain rational parameterizations of the tube circle (or the outer circle). Of course, we also get them by taking the limit of P when $u \to \infty$ ($v \to \infty$).

Intuitively, this parameterization cuts the cyclide along the outer circle and the tube circle, and "rolls out" the cyclide to the plane. Therefore, we call the outer circle and the tube circle the *cut circles* of the cyclide.

We also use the homogeneous parameterization of the cyclide, where the denominator is written as a separate variable:

$$\hat{P} : \mathbb{R}^2 \to \mathbb{R}^4, \begin{pmatrix} u \\ v \end{pmatrix} \mapsto \begin{pmatrix} \mu(c(1+u^2)(1+v^2) - a(1-u^2)(1-v^2)) + b^2(1-u^2)(1+v^2) \\ 2u(a(1+v^2) - \mu(1-v^2))b \\ 2v(c(1-u^2) - \mu(1+u^2))b \\ a(1+u^2)(1+v^2) - c(1-u^2)(1-v^2) \end{pmatrix}.$$

Here are the homogeneous parameterization for the tube circle

$$\hat{PT} : \quad \mathbb{R} \to \mathbb{R}^4, \quad v \mapsto \begin{pmatrix} \mu(c(1+v^2) + a(1-v^2)) - b^2(1+v^2) \\ 0 \\ -2v(c+\mu)b \\ a(1+v^2) + c(1-v^2) \end{pmatrix}$$

and the outer circle

$$\hat{PO}: \quad \mathbb{R} \to \mathbb{R}^4, \quad u \mapsto \begin{pmatrix} \mu(c(1+u^2)+a(1-u^2))+b^2(1-u^2) \\ 2u(a+\mu)b \\ 0 \\ a(1+u^2)+c(1-u^2) \end{pmatrix}.$$

We will also use the following homogeneous representation of the pole. Note that $\hat{p}$ indeed represents p, since $b^2 = a^2 - c^2$:

$$\hat{p} := \begin{pmatrix} -\mu(a-c)-b^2 \\ 0 \\ 0 \\ a-c \end{pmatrix}.$$

5.3.3. Our implementation

GeometryTraits: We aim to represent the curves on the cyclide as algebraic curves in parameter space, and compute the arrangement of these plane curves. Let P denote the parameterization of the cyclide. Consider a surface of degree n, implicitly defined by $F = \sum_{i,j,k} a_{ijk} x^i y^j z^k \in \mathbb{Z}[x,y,z]$, and let $\hat{F} = \sum_{i,j,k} a_{ijk} x^i y^j z^k w^{n-i-j-k}$ denote its homogenization.

Lemma 5.3.2. *The vanishing set of $f := \hat{F}(\hat{P}(u,v)) \in \mathbb{Z}[u,v]$ parameterizes the intersection points of F with the cyclide away from the cut circles.*

Proof. By definition, the vanishing set of $F(P(u,v))$ in $\mathbb{R}^2$ defines the intersection curve of F and P away from the image of the boundary, which are exactly the cut circles. On the other hand, $F(P(u,v)) = 0$ if and only if $f = \hat{F}(\hat{P}(u,v)) = 0$. □

Given algebraic surfaces $V(F_1), \ldots, V(F_m)$ with $F_i \in \mathbb{Z}[x,y,z]$ (of arbitrary degree), Lemma 5.3.2 yields polynomials $f_1, \ldots, f_m$ whose vanishing sets are the intersection curves with the cyclide in its parameter space. Thus, to compute their arrangement on the cyclide, we can simply use our model for the `ArrangementTraits_2` concept for arbitrary algebraic plane curves, as described in Chapters 3 and 4. Figure 5.6 shows an example of such an arrangement. This already answers the question about the `GeometryTraits_2` model, as described in Section 5.3.1.

We point out that, although representing the intersection curves explicitly in parameter space seems to be the most natural approach for performing a sweep in the parameter space, the cyclide is actually the first case of a reference surface where this approach is taken. For the case of great circles on the sphere, one can exploit the fact that great circles are defined by a plane through the origin, and deduce the geometric predicates directly from the planes. For elliptic quadrics, the sweep is also only simulated by projecting all curves to the plane and deducing the predicates from the arrangement in the projection. We refer the reader to [BFH$^+$09a] for any more details.

Note that the degree of the intersection surfaces $V(F_1), \ldots, V(F_m)$ is, in principle, not restricted. However, the curves f_i are of bi-degree up to $(2 \cdot \deg F_i, 2 \cdot \deg F_i))$,[51] which limits the usability of our implementation for surfaces of high degrees.

[51] A different parameterization of the cyclide might lead to curves f_i of smaller (bi)-degree, We are neither aware of a better parameterization, nor of a result that proves optimality of the chosen $\hat{P}$.

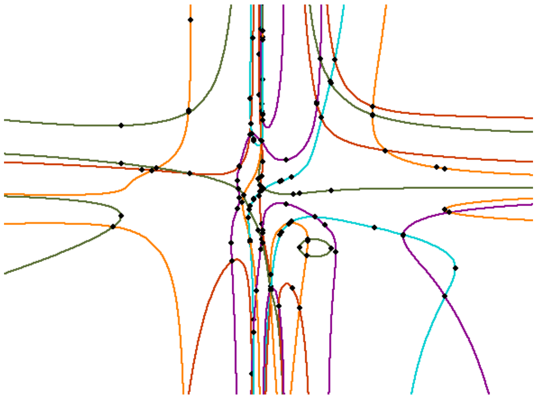


Figure 5.6. Cut-out of an arrangement in the parameter space of a cyclide, induced by 5 intersecting surfaces of degree 3.

TopologyTraits: We need a special treatment for unbounded segments in the parameter space. They correspond to segments on the cyclide that intersect the cut circles. Recall that in the plane, we compute a formal endpoint for each unbounded segment which is either one of the four combinations $(\pm\infty, \pm\infty)$, or some point $(\alpha, \pm\infty)$, or some point $(\pm\infty, \beta)$ (for this last case, we need to extend the curve analysis by the method of detecting horizontal asymptotic arcs, as explained on page 129).

One can verify that $P(\alpha, +\infty) = P(\alpha, -\infty) = PO(\alpha)$ and $P(+\infty, \beta) = P(-\infty, \beta) = PT(\beta)$, where PO and PT are the parameterizations of the outer circle and the tube circle, respectively. Moreover, $P(\pm\infty, \pm\infty) = p$. The strategy for unbounded segments is thus as follows: We maintain two sequences of DCEL vertices that store the already created vertices for points of the form $(\alpha, \pm\infty)$ and $(\pm\infty, \beta)$. The sequences are called the u-sequence and the v-sequence. Each vertex knows the u- (or v-)coordinate that it is assigned to. Whenever an unbounded segment with formal endpoint $(\alpha, \pm\infty)$ or $(\pm\infty, \beta)$ appears during the arrangement construction, we check whether a DCEL vertex for this point already exists in the u- or v-sequence. If yes, the existing DCEL vertex is used, otherwise, a new DCEL vertex is created at this position. For the pole, we can proceed in a similar way, considering unbounded segments with formal endpoint $(\pm\infty, \pm\infty)$.

This strategy, in particular, handles the identification of opposite boundary sides, because it does not distinguish between $(\alpha, +\infty)$ and $(\alpha, -\infty)$ (and the same for $(\pm\infty, \beta)$). However, only considering the unbounded segments from the interior of the parameter space does not always suffice to construct the correct arrangement on the cyclide. We give two examples:

- Assume that one of the surfaces touches the cyclide at a point on a cut circle (and that this point is not part of any other surface of the input). In this case, the arrangement should have an isolated vertex at this position. However, in the parameter space, this isolated point lies at infinity and is simply not detected when only processing the arrangement induced by $V(f_1), \ldots, V(f_m)$, since no segment is adjacent to it.
- Assume that one of the surfaces intersects the cyclide in a whole cut circle. This means that there is a vertical (or horizontal) line at infinity in the parameter space. By only considering the arrangement $V(f_1), \ldots, V(f_m)$, this line is not detected and DCEL edges between the vertices at infinity are not created.

It is relatively simple to extend the algorithm so that these special cases are also correctly handled. However, these cases cannot be covered by the current design of the `Arrangement_on_surface_2` package and are thus not implemented. The integration is planned for a future release.

The strategy is as follows: For a surface $V(F_i)$ of total degree n, we consider $\hat{F}_i(\hat{PO}(u)) = \operatorname{coef}(f_i, v, 2n)$. If this is zero, the whole outer circle is in the intersection, and we store this information for a post-processing step. Otherwise, if $\operatorname{coef}(f_i, v, 2n)$ is a univariate polynomial, we isolate its real roots. Each such root α corresponds to an intersection of $V(F_i)$ and S at the outer circle; we add a DCEL vertex representing $(\alpha, \pm\infty)$ to the u-sequence (this ensures that we do not forget any isolated vertex at infinity). For the tube circle PT, we proceed analogously. Moreover, if $\hat{F}_i(\hat{p}) = \operatorname{coef}(\operatorname{coef}(f_i, v, 2n), u, 2n) = 0$, we add a DCEL vertex for the pole, if one does not yet exist. Then, we sweep the arrangement induced by $V(f_1), \ldots, V(f_m)$ as before, with the special treatment for unbounded segments. After the arrangement has been built up, we deal with possible lines at infinity. If $\operatorname{coef}(f_i, v, 2n) = 0$ for any $i \in \{1, \ldots, m\}$, we create edges between two consecutive DCEL vertices in the (sorted) u-sequence, and connect the first and last elements in the u-sequence with the pole vertex. We proceed analogously for the v-sequence, in case that $\operatorname{coef}(f_i, u, 2n) = 0$ for any $i \in \{1, \ldots, m\}$.

We have only considered the handling of DCEL vertices and edges so far. We also have to consider the problem of maintaining faces. In the DCEL construction, faces must be updated whenever an edge insertion closes a loop. Let us concentrate on the case in which the closed loop does not intersect the outer bounding cycle of the face F it was contained in. In the plane, it is rather clear what happens in this case: A hole H of F is created that has the loop as its outer bounding cycle. On surfaces with one pair of identification lines, like the sphere, there is no canonical choice of "interior", and one has some freedom to define what should be understood as a "hole". Depending on this choice, a closed loop might split a face into two faces, and none of them can be considered to be a hole of the other.

For our case of a cyclide, there is even a third possible situation after closing a loop, namely that the face might not split at all, because the Jordan curve theorem does not hold on the cyclide. This happens as soon as the first non-contractible loop is closed (which means that the loop that cannot be contracted to a single point). In this situation, the loop does not split a face, but any additional loops will (since otherwise, the genus would be two or more).

One can show that a loop is non-contractible if it intersects one line of identification (top/bottom or left/right) an odd number of times. This helps to distinguish the third case from the former two. Further cases require a more elaborate case distinction, and depend on the strategy of how to define holes on a surface. We refer the reader to [Ber08, §4.4.3] for a complete description.

5.3.4. Experimental results

We take randomly generated surfaces (by randomly choosing coefficients) and compute the arrangement induced by them on the torus defined by the parameters $a = 2, b = 2, \mu = 1$. Our implementation is capable of handling other cyclides, also those not in standard position and orientation.

Instance	onCylide	param	planar	non-opt
(2,16,10)	1.23	0.03	1.21	2.17
(3,16,10)	6.03	0.32	5.73	9.83
(4,16,10)	26.55	2.46	23.63	37.47
(5,16,10)	91.05	13.53	78.24	136.18
(6,16,10)	273.45	56.87	218.53	385.32
(3,32,10)	7.34	0.48	6.83	12.56
(3,64,10)	9.58	1.00	8.69	16.93
(3,128,10)	16.24	2.66	13.54	29.57
(3,256,10)	32.39	8.13	24.20	61.28
(3,512,10)	74.97	24.35	49.12	147.96
(3,1024,10)	183.11	71.32	111.93	382.21
(3,16,20)	23.52	0.68	22.96	31.09
(3,16,40)	94.09	1.30	92.93	109.65
(3,16,60)	209.17	1.94	206.62	232.10
(3,16,80)	370.43	2.57	367.12	401.08
(3,16,100)	571.38	3.25	568.61	611.07

Table 5.3. Benchmark results for random curves. (n, c, m) stands for m surfaces of degree n, and each coefficient randomly from the range $[-2^{c-1}+1, 2^{c-1}]$.

In Table 5.3, we list the total running times for computing the arrangement on the torus (column "onCyclide") and the time spent computing the defining polynomials of the intersection curves in the parameter space (Column "param"). The time for computing the planar arrangment induced by these curves in the unbounded plane is also listed (column "planar"). Finally, we give the computation time for the case in which the optimization of Section 4.1.4 is switched off, which means that a curve analysis is done in a sheared system as soon as a vertical asymptotic arc is detected.

We can observe that the total time for computing the arrangement on the surface equals the time to get the bivariate polynomials plus the time to compute their arrangment in $\mathbb{R}^2$.[52] Computing the defining polynomials in parameter space becomes non-negligible for higher degrees and coefficients. One could reduce this effect, since this part is currently implemented with a rather adhoc solution. Still, the main result of our experiments is that the additonal topological overhead of being on a cyclide instead of in the plane does not affect the performance of the implementation. This result remains true for other base cyclides, and still holds true if one forces degeneracies in the arrangement. We refer the reader to [BK08] [BFH+09a] for more detailed experimental results.

The last column shows the success of the additional optimization for simple asymptotic arcs as presented in Section 4.1.4. Since practically all curves in parameter space have such vertical asymptotes, the effect becomes particularly visible in this setting. The additional time is spent completely in the analyses of the single curves.

[52]The running times in each column have been measured using different test runs. Due to randomization in the algorithm (and also due to the limited precision of time measurements), the timings do not always match perfectly.

Summary

Owing to our generic software design, our arrangement algorithm can be combined with the new `Arrangement_on_surface_2` package. This yields arrangements on parameterizable surfaces, by reusing the same code as that for the planar case. We have presented the solution for the currently most challenging case, that being arrangements on ring Dupin cyclides.

5.4. Further applications

We present two more applications that use our framework for algebraic curves. Both are currently works in progress, and further underline the usefulness of the software that we provide with the algebraic kernel.

5.4.1. Boolean set operations on curved polygons

It is well-known that computing overlays of arrangements (Section 4.2.4) directly yields Boolean set operations (union, intersection, symmetric difference, etc.) of polygonal bounded regions [dBvKOS00, §2]: Given two polygonally bounded sets P and Q, represented as arrangements, we want to compute a Boolean set operation (e.g., their intersection). Each cell of the arrangement gets a flag whether it belongs to the set or not. When computing the overlay $\mathcal{O}$ of P and Q, each of its cells originates as an intersection of a cell c_P of P with a cell c_Q of Q. The flag of the cell in the overlay is computed by applying the corresponding Boolean operation (e.g., $\wedge$ in case of intersection) on the flags of c_P and c_Q. The result of the set operation is the union of all cells where the flag is set. Since our software is able to compute algebraic segments and arrangements as well as their overlay, there is no algorithmic obstacle realizing Boolean set operations for sets bounded by algebraic segments.

Cgal offers suitable and convenient data types to realize such Boolean set operations. The class template `General_polygon_set_2` internally defines types `Polygon_2` and `Polygon_with_holes_2`, which model polygons whose boundaries are x-monotone segments. The type of these segments is controlled by the traits class that is passed to the `General_polygon_set_2` as the template argument; this traits class also must provide basic geometric operations on such segments. The requirements are composed into a concept called `GeneralPolygonSetTraits_2`. In fact, the `Curve_kernel_via_analysis_2` that we provide is a model of this concept. The only additional requirement compared to the `ArrangementTraits_2` concept is that segments must have a direction in order to determine which side of the segment is inside the polygon and which one is outside.

With a proper instantiation of `General_polygon_set_2`, we can use the internally defined polygon types to apply the Cgal functions `intersection`, `join`, `complement`, and `symmetric_difference` that allow the corresponding set operations. Eric Berberich has recently completed a first example that computes the intersection of two "algebraic" polygons. We believe that there are no difficulties left on the algorithmic side, but we are still aiming for a demo program with a graphical user interface in the future.

5.4.2. Voronoi diagrams of lines in $\mathbb{R}^3$

This is current work in progress by Dan Halperin, Michael Hemmer, and Ophir Setter.[53] For completeness, we define what is understood by a Voronoi diagram.

Definition 5.4.1 (Voronoi diagram). For a set of m objects (called sites) $o_1, \ldots, o_m$ in $\mathbb{R}^n$, and $p \in \mathbb{R}^n$ arbitrary, define

$$c(p) := \{o_i \mid i \in \{1, \ldots, n\} \wedge \forall\, j \in \{1, \ldots, n\} : \|p - o_i\|_2 \leq \|p - o_j\|_2\}$$

to be the set of closest sites to p. The *Voronoi diagram* is the subdivision of $\mathbb{R}^n$ into maximally connected regions such that all points in a region have the same closest sites. The set of the subdivision that contains o_i is called the *Voronoi cell* of o_i.

The classical case is the one where the sites are points in the plane. Then, the induced Voronoi diagram is an arrangement of line segments. The reason is just that the *bisector* of two sites (the set of points that have the same distance to both sites) is a line in $\mathbb{R}^2$. The *trisectors* (the set of points that have the same distance to three sites) are consequently single points, the intersections of two bisectors.

For lines in $\mathbb{R}^3$, the situation is more complicated. A bisector is defined by a hyperbolic paraboloid, that is, an algebraic surface of degree 2. Trisectors are the intersections of two such surfaces, namely algebraic space curves of a degree of up to 4. Halperin et al. compute a representation of each Voronoi cell separately. Consider a cylinder around a site. Each point p on the cylinder defines a unique ray starting at the site and orthogonal to it. Assigning to p the first bi- or trisector that is hit by that ray induces an arrangement on the cylinder (this is also called the *lower envelope* of the line) that represents the Voronoi cell. This arrangement can be obtained by projecting the trisectors, which leads to algebraic curves on the cylinder.

Instead of working directly with the cylinder, the authors choose a projection plane that touches it, and project the trisectors to this plane. The advantage is that the degrees of the projected trisectors are of degree at most 8. Their arrangement can again be computed with our algebraic kernel. Further details are skipped here, but we observe that algebraic curves of high degrees arise quite directly, even when considering linear input objects.

[53] The author thanks all three for making their current status of research available.

I believe in the third dimension but not in the other two.

James Parry

6 Stratification and Triangulation of Algebraic Surfaces

Let us now transfer our ideas from $\mathbb{R}^2$ to $\mathbb{R}^3$ and discuss the analogue of the curve analysis (Section 3.2) for algebraic surfaces of arbitrary degree: recall that the curve analysis can be seen as a method to "discretize" an algebraic curve, that is, to compute an embedded graph isotopic to the curve. For the case of algebraic surfaces, this corresponds to a piecewise linear mesh isotopic to the surface. We will compute such a triangular mesh, with the extra property that all vertices of the mesh are lying on the surface, and arbitrarily close approximations are possible.

The results presented in this chapter have appeared in [BKS09]. Parts of it have also been covered in [BKS08] and in [Ber08, §5]. We proceed in two steps. First, we compute a cell decomposition of the surface, called *stratification* of the surface (see [BPR06, §5.5] and compare also the similar notion of a CW-complex [Mas67] [Bre95]):

Definition (stratification). Let S be a surface. A *stratification* of S is a decomposition of S into cells such that:

- each cell is a smooth subvariety of S of dimension 0, 1, or 2.
- it has the *boundary property*, that is, the boundary of a cell is given by a union of other cells.

The cells of a stratification are also called *strata*.

The stratification that we compute has the additional property that each cell is xy-monotone, which means that any vertical ray hits a cell at most once (except for vertical strata), and the xy-ranges of two cells either coincide or are completely disjoint. This yields a cylindrical stratification, and is the analogue of the segmentation of algebraic curves discussed in Section 2.2.3. Apart from the decomposition, we also compute how the cells are connected, that is, we compute the boundary of each cell.

One well-known example of such a stratification is induced by the cylindrical algebraic decomposition ,with respect to the surface, of $\mathbb{R}^3$. Our solution consists of only $O(n^5)$

cells (with n being the degree of the surface), whereas the worst-case complexity of a *cad* is $O(n^7)$. This improvement is due to the fact that we first project the critical points of the surface into the xy-plane, as in the classic *cad* algorithm does, but we do not apply the *cad* scheme recursively, and instead restrict to the planar arrangement induced by the projection, It is also possible to refine the decomposition into simply connected cells without increasing the asymptotic complexity of the decomposition.

The stratification algorithm essentially uses the same toolbox of computational methods as the curve analysis algorithm from Section 3.6 does; for instance, resultants for the projection from $\mathbb{R}^3$ to $\mathbb{R}^2$, the bitstream-Descartes method (and its m-k extension) to compute the fiber at points in the xy-plane, and interval arithmetic, for example during the adjacency step to compute the cell connections. It should be clear at this point that we aim for an exact result in all cases, and our algorithmic design is guided by the idea of avoiding expensive symbolic computation as much as possible.

Despite all these similarities to the curve analysis, the stratification algorithm is far from being "analogous" to the curve case. A main difference is that the stratification is computed without shearing the surface, even in degenerate examples. One reason is that the scheme of shearing, analyzing the sheared objects, and finally shearing back seems not to be easily transferable from curves to surfaces. Also, we have mentioned the general disadvantages of shearing, particularly in Section 4.1.4. However, we pay a price to avoid to shearing, since we have to deal with any kind of degenerate situation in the original system. The most difficult degeneracy to handle are surfaces with vertical line components, mainly because they lead to a fiber with infinitely many points at some position. For planar curves, such lines can be factorized out and treated separately (Section 3.2.4), but algebraic surfaces might support such lines even if they are irreducible.

The stratification reveals topological information about the surface, but it does not give the complete topology in the sense that a simplicial complex isotopic to the surface is not immediately deducible. Therefore, we have to turn the stratification into a triangulation in a second step. This requires computing the *cad* of the surface with its adjacency information, which is an easy task using the adjacency information of the stratification. The triangulation consists of up to $O(n^7)$ cells. For unbounded surfaces, a three-dimensional axis-aligned bounding box which contains all relevant features of the surface is computed first, and the triangulation is restricted to this box.[54]

EGC implementations of both the stratification algorithm and the triangulation algorithm are provided. They both make extensive use of the ability to compute and manipulate arrangements of curves with arbitrary degree, using the `Algebraic_kernel_2` package presented in Section 4.2.1. For a surface of degree n, the degree of the projected curves under consideration grows to $n(n-1)$, which in practice restricts the algorithm to surfaces of small degree. However, as we demonstrate, interesting surfaces from algebraic geometry can be analyzed with our approach.

As mentioned, our triangulation consists of $O(n^7)$ cells and constitutes a stable isocomplex for the surface. By a simple post-processing step, we can obtain a general isocomplex for the surfaces that has complexity of $O(n^5)$, the same as for our stratification. We also discuss whether the topology of the surface can be described with fewer triangles. We construct an example surface, for which $\Omega(n^4)$ triangles are necessary for a stable isocomplex. In the general case, $\Omega(n^3)$ triangles are necessary in the worst case. For curves, we can

[54]The algorithm would also support computing the triangulation in a user-defined box.

prove tight bounds for the same problem.

Related work The problem of topology computation for algebraic plane curves has been extensively studied; see the related work section of Chapter 3 for an overview. Exact methods for the case of space curves [Kah08] [DMR08] [AS05] [GLMT05] have also been considered.

On the problem of topology computation for algebraic surfaces, two principle approaches can be distinguished: one is to consider *level curves* of the surface for certain critical values and to connect the components of these levels in order to obtain a topological description of the surface; see the recent work of Alberti et al. [AMT09] (also in [BCSM$^+$06]), Fortuna et al. [FGL04] [FGPT03] (for non-singular curves) and Alcázar et al. [ASS07] (where the connection step is missing). The other approach is to project the critical points of the surface to the plane, obtaining the *silhouette curve*. The topology is then deduced by lifting the arrangement cells induced by the silhouette. We are following this strategy; see also Cheng et al. [CGL05] and the articles about *cad* below.

The tools to compute a surface's topology are similar in all approaches mentioned: each one needs to compute the topology of algebraic plane curves to analyze either the level curves or the silhouette. Additionally, critical points of the surface, or at least their projections, must be identified, which is usually done by resultant calculus or Groebner bases. Most algorithms, for example, [AMT09] [FGL04] [FGPT03] [CGL05], apply a linear (topology-preserving) transformation to obtain a generic (or at least normal) position that simplifies the computation. As already said, we decided not to allow such a transformation in our algorithm in order to also preserve geometric properties of the surface.

None of the articles [AMT09] [FGL04] [FGPT03] [ASS07] [CGL05] reports on the practical performance of their algorithms; if implementations are mentioned at all,[55] they mainly propose carrying out the calculations symbolically, or leaving the concrete implementation of certain substeps open.

We have already mentioned in Section 1.2 that computing a *cad* with adjacency information is closely related to our problem. Some ideas behind our algorithm have already appeared in similar form in this context: Improvements to the projection step reduce the number of polynomials considered in the *cad* [Bro01] [McC98], and cells in the *cad* are combined into clusters to reduce the complexity [Arn88]. We discuss the similarities and differences with the appropriate references when we discuss the algorithm in detail.

Outline of this chapter We first investigate the stratification algorithm, namely, what sort of arrangement is computed in the projection plane (Section 6.1), how to obtain the fiber of any cell in the arrangement (Section 6.2), and how to compute the adjacencies between the cells (Section 6.3). Then, we turn to the triangulation of surfaces (Section 6.4), and discuss the case of compact surfaces first (Section 6.4.1), and the general case subsequently (Section 6.4.2). Experimental results are presented in Section 6.5. Finally, we give lower and upper bounds on the output complexity of isocomplex computations, stable or general, in Section 6.6.

[55] Complete implementations have been presented for subclasses of surfaces, such as intersections of quadrics [BHK$^+$05] and meshes of non-singular surfaces [PV07].

6.1. The n-k-arrangement

An algebraic surface is the vanishing set of a trivariate polynomial $f \in \mathbb{Z}[x,y,z]$. In what follows, we always write $S := V(f)$ with $f \in \mathbb{Z}[x,y,z]$ for an algebraic surface of degree n, and we set $n_z := \deg_z(f)$. We henceforth assume that f is a square-free and primitive polynomial, that is, S contains no irreducible component twice, and has no two-dimensional vertical component. The treatment of non-primitive polynomials consists of a separate analysis of the primitive part and the vertical part, similar to the vertical line case for algebraic curves.

We will first compute an arrangement in $\mathbb{R}^2$, called the *n-k-arrangement*, containing certain information about the z-fiber $V(f(\alpha,\beta,z))$ at a point $(\alpha,\beta) \in \mathbb{R}^2$. To define the arrangement, we need the following definition:

Definition 6.1.1 (local degree, local gcd degree, local real degree). For $p = (\alpha,\beta) \in \mathbb{R}^2$, we write $f_p := f(\alpha,\beta,z) \in \mathbb{R}[z]$ and

$$\begin{aligned} n_p &= \deg(f_p) \\ k_p &= \deg_z(\gcd(f_p, f_p')) \\ m_p &= \#\{z_0 \in \mathbb{R} \mid f_p(z_0) = 0\} \end{aligned}$$

where n_p is called the *local degree*, k_p the *local gcd degree*, and m_p the *local real degree.*

Definition 6.1.2 (n-k-invariance). A connected set C is *n-k-invariant* if there exist numbers n_C and k_C such that for each $p \in C$ the local degree is n_C, and the local gcd degree is k_C.

Definition 6.1.3 (n-k-arrangement). An *n-k-arrangement* is an arrangement where each cell is n-k-invariant.

In his seminal paper about cylindrical algebraic decomposition, Collins [Col75] has proved that f is delineable over any n-k-invariant set, that is, that the (real) lift over the set is the union of m disjoint function graphs. We state a slightly weaker version of his theorem (note also that it generalizes Theorem 2.2.10 (Delineability Theorem)):

Theorem 6.1.4. *Let C be an n-k-invariant set. Then, each $p \in C$ has the same local real degree m_C. Moreover, for each $i = 0, \ldots, m_C$, the i-th lift*

$$C^{(i)} := \{(p_x, p_y, z_i) \in C \times \mathbb{R} \mid z_i \textit{ is the i-th distinct root of p's z-fiber}\}$$

over C is connected.

Proof. Over an n-k-invariant set, the number of distinct complex roots is constantly $n_C - k_C$. The roots of $f(p,z)$ continuously depend on p, thus, in an open neighborhood of any point of C, the imaginary roots stay imaginary. As the total number of roots is preserved and an imaginary root only appears together with its complex conjugate, the real roots also remain real; see [Col75, Thm. 1] for more details. □

The next construction also appears in Collins' work [Col75, Thm. 4].

Definition 6.1.5 (truncation). For $f = \sum_{i=0}^{n_z} a_i(x,y)z^i \in \mathbb{Z}[x,y,z]$, we denote the truncated polynomial of degree j by

$$f_j := \sum_{i=0}^{j} a_i(x,y)z^i \in \mathbb{Z}[x,y,z]$$

for $j = 0,\dots,n$.

Theorem 6.1.6. *For each algebraic surface S, there exists an n-k-arrangement.*

Proof. We give a constructive proof. Let p be an arbitrary point in the plane and $f = \sum_{i=0}^{n_z} a_i(x,y)z^i$. The local degree of f at p simply depends on the "coefficient polynomials" $a_n,\dots,a_0 \in \mathbb{R}[x,y]$ by

$$n_p = \deg f_p = \max_{i=0,\dots,n_z} \{i \mid a_i(p) \neq 0\}.$$

In the same way, the local gcd degree depends on the principal Sturm-Habicht coefficients $\mathrm{stha}_i(f_{n_p}) \in \mathbb{R}[x,y]$ by

$$k_p = \deg\gcd(f_p, f_p') = \min_{i=0,\dots,N_z} \{i \mid \mathrm{stha}_i(f_{n_p})(p) \neq 0\}$$

(this follows from Lemma 2.3.14 and the specialization property (Theorem 2.3.17)). The a_i's and $\mathrm{stha}_i(f_{n_p})$'s define plane curves $\alpha_i = V(a_i)$ and $\sigma_{j,i} = V(\mathrm{stha}_i(f_j))$, respectively, of a degree of at most $n(n-1)$. The values n_p and k_p are determined by the curves that p is part of. It follows that the arrangement induced by $\alpha_{n_z},\dots,\alpha_0$ and for all $j = 1,\dots,n_z$, $\sigma_{j,0},\dots,\sigma_{j,j}$ has only n-k-invariant cells. ☐

The proof presents a way to compute an n-k-arrangement for a surface. However, the resulting arrangement consists of many more cells than actually necessary – we aim for an n-k-arrangement consisting of fewer cells:

Definition 6.1.7 (silhouette). The *silhouette* Γ_S of S is defined by $\mathrm{stha}_0(f) = \mathrm{res}_z(f, \frac{\partial f}{\partial z}) \in \mathbb{Z}[x,y]$.

Lemma 6.1.8. *For any point, $(n_p,k_p) = (n_z,0)$ if and only if p is not on Γ_S. As a consequence, all edges and vertices of an n-k-arrangement away from Γ_S can be merged with their adjacent faces to an n-k-invariant face.*

Proof. Using [BPR06, Prop. 4.27], we have $\mathrm{res}_z(f, \frac{\partial f}{\partial z}) = a_{n_z}\mathrm{disc}(f)$ where $\mathrm{disc}(f)$ denotes the discriminant of f. Clearly, $n_p = n_z$ for a point p if and only if $a_{n_z}(p) \neq 0$. From the definition of the discriminant, $k_p = 0$ for a regular point p if and only if $\mathrm{disc}(f)(p) \neq 0$. ☐

Having any n-k-arrangement, we can turn it into a minimal n-k-arrangement by a post-processing step (we assume that each arrangement cell C stores the numbers n_C and k_C as data): Remove all edges and vertices away from Γ_S, and remove vertices on Γ_S that have exactly two adjacent edges if both edges have the same local degree and local gcd degree as the vertex. Merge the adjacent edges in this case. We next present an algorithm that integrates this post-processing step in the arrangement computation, in order to lower the size of the intermediate arrangements in the algorithm. The main tool is the computation of overlays (Section 4.2.4). We start by computing the arrangement A defined by only the silhouette Γ_S. Each face get the values $(n, 0)$ according to Lemma 6.1.8. We first decompose A such that each cell has an invariant local degree. To do so, repeat the following steps for $j = n_z, \ldots, 0$: Overlay A with the arrangement of the curve α_j; the result is A'. Remove vertices and edges of A' that lie on a face of A. Also, remove all vertices of A' that lie on an edge of A whose local degree has already been set. For each cell that lies on a face of α_j, and whose degree is not set yet, set its local degree to j. Set $A \leftarrow A'$ and proceed with the next iteration. At the end, set the local degree of all cells which are not yet set to $-\infty$, since above these cells, S is vertical.

Next, we further decompose A into n-k-invariant cells. For that, we iterate over the degrees and overlay with the corresponding principal Sturm-Habicht coefficient curves $\sigma_{j,i}$. Repeat for $j = n_z, \ldots, 1$: Repeat for $k = 0, \ldots, j-1$: Overlay A with the arrangement of $\sigma_{j,k}$; the result is A'. Remove vertices and edges of A' that lie on a face of A. Remove all vertices of A' that lie on an edge of A whose local gcd degree has already been set, or whose local degree does not equal j. For each cell of A that lies on a face of $\sigma_{j,k}$, whose local degree is j, and whose local gcd degree is not yet set, set the local gcd degree to k. Set $A \leftarrow A'$ and proceed with the next iteration.

We mention the obvious optimization that for the local gcd degree, one has only to consider those degrees that appear as the local degree of at least one cell. Also, one can stop the inner iteration over the k's as soon as all cells of degree j know their local gcd degree.

The n-k-arrangement computed by the above algorithm will be called $\mathcal{A}_S$ from now on. It basically consists of the overlay of the leading coefficient curve and the discriminant curve of f (compare Lemma 6.1.8). From the overlay with the remaining α's and $\sigma_{j,i}$'s, the local degree and the local gcd degree is assigned to each cell of $\mathcal{A}_S$. We point out that the edges of $\mathcal{A}_S$ do not necessarily correspond to x-monotone segments of the curve. In fact, it is even possible that an edge forms a loop, and hence has no endpoint. This is not a problem theoretically, and we can also model such n-k-edges in practice by a data type that stores a sequence of (adjacent) x-monotone segments. We remark that similar ideas have been introduced to reduce the number of cells of a *cad*. Arnon [Arn88] has proposed merging *sign-invariant* cells of a *cad*, but our notion of n-k-invariance is a strictly weaker condition and thus produces larger cells. Moreover, Brown [Bro01], based on work by McCallum [McC98], has shown that considering the leading coefficient and the discriminant are sufficient to ensure delineability. So, the consideration of the non-leading coefficients and the principal Sturm-Habicht coefficients is not necessary to ensure the statement of Theorem 6.1.4. Still, knowledge about the local degree and local gcd degree of each cell of $\mathcal{A}_S$ allows the application of fast methods in the lifting step, as we discuss in Section 6.2.

The complexity of our n-k-arrangement $\mathcal{A}_S$ is not greater than that of Γ_S.

Theorem 6.1.9. *The number of cells of $\mathcal{A}_S$ is $O(n^4)$.*

Proof. Since arrangements induce planar graphs, it is enough to count vertices. The silhouette Γ_S is of degree $O(n^2)$, so it has, by Bézout's theorem, $O(n^4)$ critical points. We have to show that the subdivision with respect to the remaining curves in the algorithm does not introduce more than $O(n^4)$ new vertices.

Consider the decomposition of Γ_S into irreducible components $\Gamma_{S,i}$ with degree ν_i, and fix one $\gamma = \Gamma_{S,i}$ of degree ν. During the algorithm, new vertices for γ (that are not removed in the same iteration) are introduced in only two iteration steps:

- first, when a coefficient curve α_j does not contain the whole curve γ. This introduces at most $\nu \cdot n$ many vertices. All further coefficient curves $\alpha_{n-1}, \ldots, \alpha_0$ do not introduce new vertices on γ, since the local degree of all edges for γ is set to n.
- second, new vertices are introduced when a Sturm-Habicht polynomial $\mathrm{stha}_k(f_j)$ does not contain the whole curve γ. This introduces at most $\nu \cdot n^2$ many new vertices. All further Sturm-Habicht curves $\mathrm{stha}_{k-1}(f_j), \ldots, \mathrm{stha}_0(f_j)$ do not introduce new vertices on γ, since the local gcd degree of all edges for γ is set to k.

After all, each $\Gamma_{S,i}$ gets at most $O(\nu_i \cdot n^2)$ new vertices, and the ν_i sum up to n^2. $\square$

For each cell C of $\mathcal{A}_S$, we pick a sample point that we denote by p_C. If C is a face, we can choose p_C to be rational. If C is an edge, we can choose the x-coordinate of p_C to be rational (or the y-coordinate, if the edge is vertical).

Extracting simply connected cells Sometimes it might be advantageous to achieve a decomposition into simply connected cells (i.e., each loop in a cell is contractible to a point). Our decomposition $\mathcal{A}_S$ does not have this property. We next propose an algorithm that transforms $\mathcal{A}_S$ into a decomposition of simply connected cells.

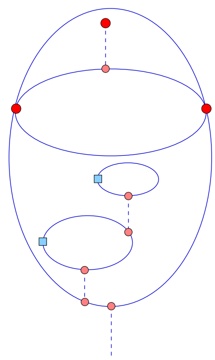


Figure 6.1. Obtaining simply connected cells for $\mathcal{A}_S$ by breaking one-dimensional circles and adding vertical arcs (dashed). Each final face is simply connected.

Only one- and two-dimensional cells of $\mathcal{A}_S$ can be non-simply connected. Consider the planar embedded graph G induced by $\mathcal{A}_S$ by mapping its 0-dimensional cells to nodes and its 1-dimensional connected cells to edges. Simple connectivity for 1-dimensional cells is achieved by adding an additional vertex to each cyclic edge; see the squared vertices in Figure 6.1.

To prevent non-simply connected faces, we apply the following algorithm: while G contains a bounded connected component, choose such a component, and connect its y-minimal point downwards using a vertical arc (dashed) until it reaches another component of G (if this does not happen, the arc goes to $-\infty$). Observe that each such arc either merges two connected components or renders one of them unbounded. Thus, the algorithm terminates, and produces a graph without bounded connected components.

The computed graph induces a refined arrangement $\mathcal{A}'_S$ of $\mathcal{A}_S$. The newly added cells inherit the n-k-properties of the cell in which they are included. For the thus refined $\mathcal{A}'_S$, we claim:

Proposition 6.1.10. *Each cell of $\mathcal{A}'_S$ is simply connected, and its number of cells is $O(N^4)$.*

Proof. Assume for the sake of a contradiction that there is a cell C of $\mathcal{A}'_S$ that is not simply connected. Clearly, C cannot be 1-dimensional because we split all cycles. So, assume that C is a face. Since it is not simply connected, there is a cycle that is not contractible. Hence, its interior contains a connected component, which must be bounded. That contradicts the fact that there is no bounded connected component.

For the complexity statement, observe that we introduce at most one edge and two vertices, and split at most one face for each connected component. Since the number of connected components is not greater than the number of faces, we add at most 4 cells for each face of $\mathcal{A}_S$. This proves that we do not increase the complexity. $\square$

We mention that this refinement into simply connected cells has not yet been integrated into our implementation that we present in Section 6.5.

Summary

The n-k-arrangement yields a cylindrical decomposition of an algebraic surface into delineable parts. Furthermore, certain information as the local degree and local gcd degree of each cell of the arrangement is gathered. This information is useful for building up the stratification.

6.2. Z-fibers and cell decomposition

We have computed a planar decomposition $\mathcal{A}_S$ in the previous step where each cell is delineable. That means that lifting a cell yields m_C disjoint parts of the surface. These parts will form the cells of our stratification. However, m_C is not known yet. Moreover, we aim for a sample point of each stratum to represent it. Thus, we want to compute the z-fibers at the sample points of $\mathcal{A}_S$.

Definition 6.2.1 (z-fiber). The z-fiber of a point $p := (p_x, p_y) \in \mathbb{R}^2$ is

$$Z_p := \{\gamma \in \mathbb{R} \mid f(p_x, p_y, \gamma) = 0\}.$$

Note that the fiber can be equal to $\mathbb{R}$, in the case where S contains the whole vertical line $\ell_p := p \times \mathbb{R}$. We aim for a method to compute the z-fiber for an arbitrary point p with algebraic coordinates in the plane, that is, isolate the real roots of the polynomial $f_p := f(p_x, p_y, z) \in \mathbb{R}[z]$. As in the planar case, computational difficulties arise because f_p might have algebraic coefficients and because f_p might have multiple roots. However, the n-k-arrangement reveals the local degree and the local gcd degree at each point, and we will use this information in our method.

In the simplest case, k_p, the local gcd degree, is zero, which means that f_p is square-free. In that case, we apply the bitstream-Descartes method (Algorithm 2.20) on f_p.

Otherwise, if $k_p > 0$, we try to use the m-k-bitstream-Descartes method (Algorithm 2.21). For that, we need to compute m_p, the number of points in the fiber. This is done using the suitable Sturm-Habicht sequence , namely $\mathrm{stha}_0(f_{n_p}), \ldots, \mathrm{stha}_{n_p}(f_{n_p})$, the sequence of principal Sturm-Habicht coefficients of the truncation of f with degree n_p. By the specialization property (Theorem 2.3.17), and by the counting property of principal Sturm-Habicht coefficients (Theorem 2.3.29), it is enough to evaluate the signs of

$\mathrm{stha}_0(f_{n_p}), \ldots, \mathrm{stha}_{n_p}(f_{n_p})$ at p to get the number of roots in the z-fiber. Note that, for this operation, we can simply apply the functor `Sign_at_2` of the `Algebraic_kernel_2` class (Section 4.2.2). This nicely demonstrates the reusability of the functions provided by our algebraic kernel. If the m-k-bitstream-Descartes method is successful, the fiber is computed.

If the method fails, we compute the square-free part f_p^* of f_p and apply the bitstream-Descartes method on f_p^*. To get the square-free part, we compute the cofactors of the Sturm-Habicht polynomials $\mathrm{stha}_0(f_{n_p}), \ldots, \mathrm{stha}_{n_p}(f_{n_p})$; their sequence contains the square-free part as shown in Lemma 2.4.22. In our implementation, we use the algorithm presented in [BPR06, Alg.8.22] to compute the cofactors.

The reader might remember our discussion in Section 2.6.2 about the disadvantages of taking the square-free part for root isolation: it might be considerably more complicated than the original polynomial itself, and even its computation is involved for polynomial with algebraic coefficients. Both arguments remain valid, but their practical effect is limited for the surface case – since our approach is restricted to surfaces of small degree (typically, surfaces of a degree of at most 5 are feasible in practice), computing the Sturm-Habicht coefficients with cofactors is not too expensive, and the root isolation can cope with the square-free part, since the polynomial degree is small enough. The analysis of the silhouette curve remains the bottleneck in the computation, which is for instance of degree 20 for a surface of degree 5.

We define the stratification of S, at least for the case where all z-fibers are finite.

Definition 6.2.2. Let S be a surface without vertical component, $\mathcal{A}_S$ the n-k-arrangement as in Definition 6.1, and m_C the local real degree of a cell $C \in \mathcal{A}_S$. The *stratification* Ω_S is defined as

$$\Omega_S := \bigcup_{C \in \mathcal{A}_s} \left(\bigcup_{i=1,\ldots,m_C} \{C^{(i)}\} \right).$$

We represent each cell of Ω_S by a sample point, obtained by lifting the sample point of its projected cell in $\mathcal{A}_S$.

Corollary 6.2.3. *For a surface of degree n without a vertical line, the number of cells in Ω_S is $O(n^5)$.*

Proof. Each z-fiber has up to n points, and the complexity of $\mathcal{A}_S$ is $O(n^4)$. □

This means that we achieve a topological description of the surface using $O(n^5)$ sample points. This is less compared to *cad*, which consists of $\Theta(n^7)$ cells in the worst case, due to its vertical decomposition strategy in the plane. However, it also provides less topological information – in particular, edges cannot always be replaced by straight-lines without changing the topology.

Clearly, one obtains a decomposition into simply-connected cells when making all cells of $\mathcal{A}_S$ simply-connected, as described in the previous section.

In Section 6.3.3, we extend Ω_S to surfaces with vertical lines; the extension still keeps the same worst-case complexity of $O(n^5)$.

Summary

Using the information collected in the n-k-arrangement, we can use the bitstream-Descartes method to compute the lift of each cell in the n-k-arrangement. Using the m-k-bitstream-Descartes method, many z-fibers can be computed quickly, and the computation of the square-free part is only needed in degenerate situations.

6.3. Cell adjacencies

According to Definition 6, a stratification fulfills the *boundary property*, that is, the boundary of each cell should be the union of other cells. Equivalently, for any two cells M_1, M_2 with $\dim M_1 < \dim M_2$, we must have $M_1 \cap \overline{M_2} = \varnothing$ or $M_1 \subset \overline{M_2}$. In the latter case we call M_1 and M_2 *adjacent*. Then the adjacency relation of such a pair can be checked at an arbitrary point $p \in M_1$, that is, the two cells are adjacent if and only if $p \in \overline{M_2}$. Theorem 6.3.1 shows that in the case of a surface S which contains no vertical line, the decomposition Ω_S given in Definition 6.2.2 already has this boundary property, thus, Ω_S is indeed a stratification.

Theorem 6.3.1. *Let $M_1, M_2 \in \Omega_S$ with $\dim M_1 < \dim M_2$ and $C_1, C_2 \in \mathcal{A}_S$ be their corresponding projections onto the plane. If C_1 has local degree $n_{C_1} \neq -\infty$ and $M_1 \cap \overline{M_2} \neq \varnothing$, then $M_1 = \overline{M_2} \cap (C_1 \times \mathbb{R})$.*

Proof. We assume that C_1 and C_2 are adjacent in $\mathbb{R}^2$, otherwise the statement is trivial. Let M_2 be the j_0-th lift of C_2 and $p = (p^*, z_0) \in \overline{M_2} \cap (C_1 \times \mathbb{R})$ be an arbitrary point, contained in a lift $C_1^{(i_0)}$ of C_1. For the lifts $p^{*(i)}$ of p^* we choose a box neighborhood B_{p^*} of p^* and also disjoint boxes $B_1, \ldots, B_{m_{C_1}}$ lying above B_{p^*} with $B_i = B_{p^*} \times \left[p^{*(i)} - \delta, p^{*(i)} + \delta\right]$ and a $\delta > 0$. We can assume that B_{p^*} and δ are chosen such that the i-th lift of $C_1 \cap B_{p^*}$ is contained in B_i. For B_{p^*} small enough, it follows that the j_0-th lift of $B_{p^*} \cap C_2$ is also contained in B_{i_0} because $p \in B_{i_0} \cap \overline{M_2}$. As a direct consequence $((B_{p^*} \cap C_1) \times \mathbb{R}) \cap \overline{M_2}$ is the i_0-th lift of $(B_{p^*} \cap C_1)$. Now, for any two points p_1^* and p_2^* on C_1, there exists a compact path Σ on C_1 that connects them. Then we consider an open covering of Σ with local neighborhoods $B_{p'}$, $p' \in \Gamma$ such that $((B_{p'} \cap C_1) \times \mathbb{R}) \cap \overline{M_2}$ is the $i_{p'}$-th lift of C_1. Then from restricting our analysis to a finite partial covering, it follows that $i_{p'} = i_0$ for all p', thus, $C_1^{(i_0)} = \overline{M_2} \cap (C_1 \times \mathbb{R})$. Now $M_1 \cap \overline{M_2} \neq \varnothing$ exactly if $M_1 = C_1^{(i_0)}$. □

6.3.1. Edge-face adjacencies

Let E be an edge of $\mathcal{A}_S$, and let F denote an adjacent face in the arrangement $\mathcal{A}_S$. We want to compute the adjacencies between cells above E and cells above F. It suffices to restrict to the lifts of the sample point $p_E = (p_x, p_y) \in E$ and to find out their adjacencies to the lifted surface patches of F. Recall that p_E is chosen such that p_x (or p_y, for vertical segments) is rational. If the local degree over p is n_z and the z-fiber over p has been computed using the m-k-bitstream-Descartes method (compare Section 6.2), adjacencies are computed similarly to the branch number computation in the curve analysis, described in Section 3.2.1: All roots but one of f_p are simple and the cells over E to which they belong have precisely one adjacent surface patch over F. The remaining surface patches must be adjacent to the distinguished root that is returned by the m-k-bitstream-Descartes method.

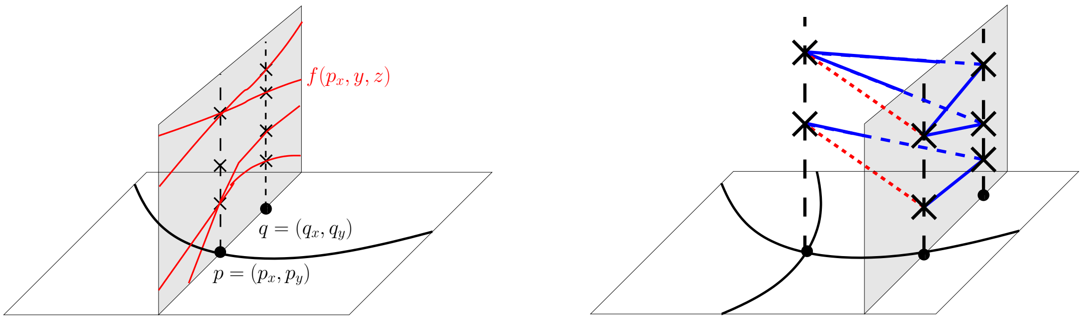


Figure 6.2. On the left: The general method for edge-face adjacencies. On the right: Illustration of the filter for vertex-edge adjacencies.

If f_p was not isolated using the m-k-bitstream-Descartes method, the treatment is the same as in [ACM88]. We choose a rational sample point $q = (q_x, q_y)$ for F with $q_x = p_x$, and consider the planar curve $f|_{x=p_x} := f(p_x, y, z) \in \mathbb{Q}[y, z]$. The i-th lift $F^{(i)}$ of F is adjacent to the j-th lift $E^{(j)}$ of E if and only if there is a segment of the curve $V(f|_{x=p_x})$ connecting the i-th point over q_y with the j-th point over p_y. Clearly, our curve analysis algorithm is used to compute the adjacency information for $V(f|_{x=p_x})$ (Figure 6.2 (left)).

6.3.2. Adjacencies of a vertex

We consider a vertex p[56] whose z-fiber is finite, let $(p_x, p_y, z_1), \ldots, (p_x, p_y, z_m)$ denote the points in its z-fiber.

If $n_p = n_z$, and p's z-fiber has been constructed using the m-k-bitstream-Descartes method, the adjacencies are computed as described in Section 6.3.1. Second, adjacencies between p and an edge E can often be derived by a transitivity argument from the combination of adjacencies of E with its adjacent faces F_1 and F_2, and the adjacencies of F_1 and F_2 to p (compare the picture on the right of Figure 6.2). We skip further details of this simple argument.

If none of these simple methods applies, choose rational intermediate values $q_0, \ldots, q_m$ such that $q_{i-1} < z_i < q_i$ for all $i = 1, \ldots, m$. The planes $z = q_i$ divide the real space in $m + 2$ *buckets* that separate the fiber points z_i.

Definition 6.3.2 (bucket-faithful). Let $C \in \mathcal{A}_S$ be adjacent to p. A point p' on C is *bucket-faithful* if there exists a path from p' to p on C such that on that path, each cell $C^{(i)} \in \Omega_S$ over C remains in the same bucket.

With a bucket-faithful point p' on C, the adjacencies of cells over C with cells over p follow by considering the z-fiber of p': if the i-th point over p' lies in the bucket of z_j, then the cells $C^{(i)}$ and $p^{(j)}$ are adjacent. Furthermore, points over p' that lie in either the bottom-most or top-most bucket belong to asymptotic components, that is, they are unbounded in the z-direction.

[56] We identify the vertex of the n-k-arrangement, and the point in $\mathbb{R}^2$ that it represents

It is easy to prove by an ε-argument that a bucket-faithful point p' exists for each cell C adjacent to p. However, we want to prevent p' being too close to p, because this results in a bad separation of the roots of $f_{p'}$ and thus slows down the computation of the z-fiber of p'.

Observe that p' on C is bucket-faithful if and only if there is a path from p' to p on C that does not intersect any of the *bucket curves* defined by $f(x, y, q_i) \in \mathbb{Q}[x, y]$. We first compute a *bucket box* around p that contains no point of any of the bucket curves (depicted on the left of Figure 6.3, the bucket curves are drawn as dashed lines). This is easily done with interval arithmetic: Use approximations of p to evaluate $f(p_x, p_y, q_i)$ for all $i = 0, \ldots, m$ until no resulting interval contains zero. The final approximation of p defines the bucket box.

In the second step, we compute bucket-faithful points inside the bucket box for each adjacent n-k-invariant cell (note that not each point inside the bucket box is bucket-faithful). For each adjacent edge, choose an arbitrary sample point, and shrink the box until all these points are outside the box (depicted in the middle of Figure 6.3). After that, each cell has a bucket-faithful point on the box boundary. Compute all intersection points of $\mathcal{A}_S$ with the box boundary.

Follow each edge E starting in p until it crosses the box boundary. The intersection point is bucket-faithful for E. For a face F, consider the edge $E \in \mathcal{A}_S$ that precedes F in counterclockwise order around p. Let p'' be the bucket-faithful point of E on the box boundary. Let p' be a point on the box boundary between p'' and the next intersection of the box's boundary with $\mathcal{A}_S$ in clockwise order. p' is a bucket-faithful point on F. The rightmost picture of Figure 6.3 shows the bucket-faithful points of the working example.

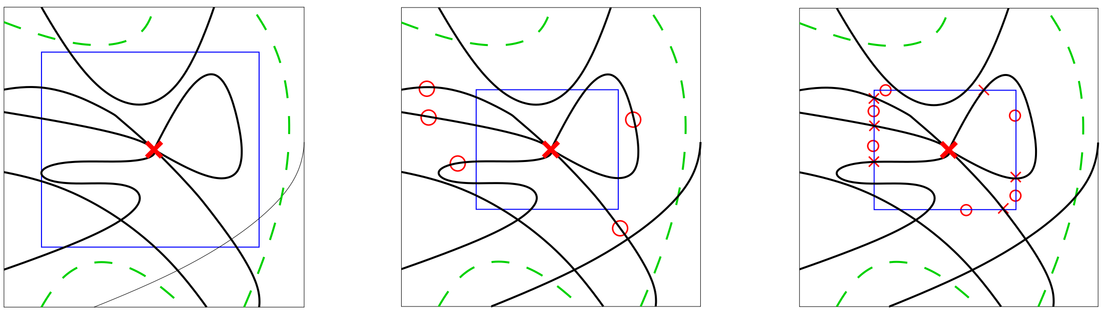


Figure 6.3. Finding bucket-faithful points: finding an initial box not containing any dashed curve (left), shrinking the box such that sample points are outside (middle), determining bucket-faithful points on the box boundary (right).

The method described has not covered the special case of an isolated vertex p. In this case, we compute the intersections of $\mathcal{A}_S$ with the vertical line $x = p_x$, and choose an intermediate value between p_y and the next intersection point above.

Our method for vertex adjacencies has a similar basic idea as the local box algorithm by Collins and McCallum [MC02] for *cads*. Still, there are some differences: our construction of the "local box" (which we call the bucket box) is more efficient since it only involves interval arithmetic. Also, we have to handle adjacent components that are not x-monotone, which complicates the computation of bucket-faithful points. Moreover, their local box algorithm requires irreducible polynomials as input, which implies a preceding factorization step.

6.3.3. Vertical lines

In the special case where S contains a vertical line ℓ_p, the lift $F^{(i)}$ of a face $F \in \mathcal{A}_S$, adjacent to p in $\mathcal{A}_S$, is, in general, no longer adjacent to exactly one lift of p, which means that the boundary property is no longer satisfied if we take the whole vertical line as one cell in the decomposition. The next theorem shows, however, that $F^{(i)}$ is adjacent to a connected set $p \times I(F^{(i)})$ on ℓ_p, that is, a single point, a line segment, a ray, or ℓ_p.

Theorem 6.3.3. *Let S contain the vertical line ℓ_p and $F \in \mathcal{A}_S$ be a face, which is adjacent to p. Then for any surface patch $F^{(j)}$ (the j-th lift of F) there exists an interval $I(F^{(j)}) \subset \mathbb{R}$, such that $p \times I(F^{(j)}) = \overline{F^{(j)}} \cap \ell_p$.*

Proof. Given two points $(p, z_0), (p, z_1) \in \overline{F^{(j)}} \cap \ell_p$, $z_0 < z_1$, on the vertical line, there exist corresponding continuous paths $\Sigma_l \subset F^{(j)}$ with $(p, z_l) \in \overline{\Sigma_l}$ for $l = 0, 1$. Now let (p, z^*) be an arbitrary point in between (p, z_0) and (p, z_1). Restricting our considerations to (end-)parts of Σ_l we can assume that for every point $(q_l, z_{q_l}) \in \Sigma_l$, we have $z_{q_0} < z^*$ and $z_{q_1} > z^*$. We now consider the projection $\Sigma_l^* \subset F$ of Σ_l onto the plane. We further denote B_ε as the open disc with radius ε and center p. Then from the definition of F, the existence of an $\varepsilon_0 > 0$ such that $\Sigma_\varepsilon := \partial B_\varepsilon \cap \overline{F}$ is connected for all $\varepsilon < \varepsilon_0$ follows. Then, Σ_ε intersects Σ_0^* as well as Σ_1^*, and because of continuity, the j-th lift $\Sigma_\varepsilon^{(j)} \subset \overline{F^{(j)}}$ of Σ_ε contains a point s_ε with z-coordinate z^*. It also follows that $F^{(j)}$ contains an arc of the z^*-level curve of S, which passes the point (p, z^*). Hence, we must have $(p, z^*) \in \overline{F^{(j)}} \cap \ell_p$. □

Theorem 6.3.3 suggests that when there are vertical lines, we still have to decompose them into segments to obtain a decomposition Ω_S of S that fulfills the boundary property. If we knew

$$Z'_p := \bigcup_{C \in \mathcal{A}_S \setminus \{p\} : p \in \overline{C}} \left(\bigcup_{i=1,\ldots,m_C} \{z_A | z_A \text{ is an endpoint of } \ell_p \cap \overline{C^{(i)}}\} \right),$$

which is the union of all endpoints of intervals $I(F^{(i)})$ and all z-values of endpoints (over p) of lifted arcs in $\mathcal{A}_S$ adjacent to p, we would simply subdivide the vertical line at these positions in order to satisfy the boundary property.

What we compute is a set Z_p^* that is a superset of Z'_p. We state the main theorem without proof in this work, and refer the reader to [BKS09] for a geometric intuition of how to get to this theorem, and for a proof of it.

Theorem 6.3.4. *Let*

$$\begin{aligned} r(x,z) &:= \operatorname{res}_y(f, f_y) = (x - p_x)^{i_0} \tilde{r}(x,z), \\ h(x,z) &:= \operatorname{res}_y(f, \operatorname{res}_z(f, f_z)) = (x - p_x)^{k_0} \tilde{h}(x,z) \end{aligned}$$

with the following definitions of exponents

$$\begin{aligned} i_0 &:= \max\{i : (x - p_x)^i | r(x,z)\}, \\ j_0 &:= \min\{j : \frac{\partial^j f}{\partial y^j}(p_x, p_y, z) \not\equiv 0\} \\ k_0 &:= \max\{k : (x - p_x)^k | h(x,z)\}. \end{aligned}$$

Then $Z_p^* := \{z|\tilde{r}(p_x,z)=0 \vee \frac{\partial^{j_0} f}{\partial y^{j_0}}(p_x,p_y,z)=0 \vee \tilde{h}(p_x,z)=0\}$ *contains all points of* Z_p'.

As a consequence of Theorem 6.3.4, we can define our stratification Ω_S in general.

Definition 6.3.5 (stratification, general case). Let S be a surface with corresponding n-k-arrangement $\mathcal{A}_S$. Let V be the set of vertices in $\mathcal{A}_S$ whose lifts are vertical lines. For $p \in V$, let ω_p denote the partition of ℓ_p into elements of Z_p^* and their induced intervals of $\mathbb{R}$. We define

$$\Omega_S := \bigcup_{C \in \mathcal{A}_s \setminus V} \left(\bigcup_{i=1,\ldots,m_C} \{C^{(i)}\} \right) \cup \bigcup_{p \in V} \omega_p.$$

By construction of Z_p^*, Ω_S has the boundary property. We can also show that vertical lines do not increase the complexity.

Theorem 6.3.6. *The number of cells of* Ω_S *is* $O(n^5)$.

Proof. Using Corollary 6.2.3, it remains to be shown that the decomposition of the vertical lines do not introduce more than $O(n^5)$ cells. The number of vertices with vertical lines is in $O(n^2)$. For a fixed p, the set Z_p^* is the union of the roots of three polynomials in z (compare Theorem 6.3.4), whose degree is at most n^3. □

Adjacencies for vertical line cells: We need to compute how the points and vertical segments on a vertical line are connected to their neighborhood. Let p denote a vertex in $\mathcal{A}_S$ having a vertical line. We proceed similarly to Section 6.3.2 by defining bucket values q_i and bucket curves $V(f(x,z,q_i)$ for each intermediate value between elements of Z_p^*. There is a complication here, because all bucket curves now are intersecting p, and we cannot build a bucket box like before using interval arithmetic. Instead, we compute the overlay of $\mathcal{A}_S$ with all bucket curves, and build a box around p that does not contain an intersection of $\mathcal{A}_S$ with any bucket curve, except at p itself (Figure 6.4 (left)).

For the sample points of edges from $\mathcal{A}_S$, we further proceed as in Section 6.3.2. Choose points at each adjacent cell of $\mathcal{A}_S$ and shrink the box until they are outside of it. Afterwards, traverse the edges starting at p and choose the first box intersection as the sample point for the edge (Figure 6.4 (right)). This point is bucket-faithful (recall Definition 6.3.2) and reveals the adjacencies between the lifted cells over the edge with the cells at the vertical line, which is correct due to the construction of Z_p^*.

For an adjacent face F, we first compute which patches $F^{(j)}$ over F are adjacent to whole vertical segments. Each such vertical segment contains one of the bucket values q_i. Thus, a patch over F that is adjacent to an interval causes an arc of the bucket curve for q_i that lies in F and ends in p. We proceed as follows. Iterate over the arcs of all bucket curves in F that leave p. Let q_i be the bucket value of the bucket curve under consideration. Choose a sample point on the bucket curve (inside the bucket box), build the z-fiber over it, and determine which patch $F^{(j)}$ has the z-coordinate q_i. Mark this patch as adjacent to the vertical segment containing q_i, and also to the two endpoints of the segment.

Finally, when all patches adjacent to an interval are detected, consider the remaining patches. They are adjacent to some zero-dimensional cell over p. Choose a bucket-faithful point for the face (analogously to Section 6.3.2) and determine the buckets which the remaining patches belong to.

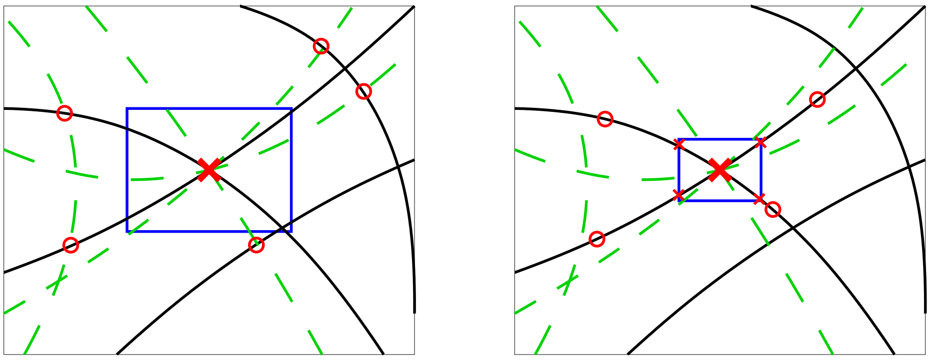


Figure 6.4. Finding bucket-faithful points in the vertical case.

Summary

The adjacency computation nicely summarizes our overall strategy of avoiding costly computations: We try to get the adjacency information combinatorially (by exploiting the output information of the m-k-bitstream-Descartes method), and only fall back to more complicated method if necessary. Simple cases (edge-face adjacencies) can be solved using efficient methods (curve analysis of a slice curve $f(q, y, z)$ with $q \in \mathbb{Q}$). Adjacencies for vertices are also solved combintatorially (by a transistivity argument) in good-natured cases. Only if this is impossible, more expensive methods are used. Even in this case, we can apply, at least partially, fast approximate methods (interval arithemtic).

6.4. Triangulation

We describe in this section how the stratification Ω_S, in combination with its adjacency information, leads to an exact triangulation $\mathcal{T}_S$, that is, an isocomplex for the surface (Definition 2.1.8). Our solution even yields a stable isocomplex, meaning the vertices of the resulting complex do not move during the isotopic transformation. This also preserves some geometric structure of the surface, since each vertex of the triangulation is a point on the surface.

The basic idea for the triangulation is as follows: The n-k-arrangement $\mathcal{A}_S$ of the surface S is transformed into an isotopic straight-line arrangement (for that, edges of $\mathcal{A}_S$ must be further subdivided), and this arrangement is refined to a planar triangulation. By the adjacency information of S, a triangulation of the surface is computed by lifting the triangles of the planar triangulation. We can show that this triangulation is isotopic to S, by constructing a stratification of S whose faces are pseudo-triangles that are in an one-to-one correspondence with the triangles of S.

A straightforward way to achieve a planar triangulation as above is to refine $\mathcal{A}_S$ into a *cad* of the plane. We note that the idea of using a *cad* to triangulate surfaces has been described already in [BPR06] in a more abstract context (compare Theorem 5.43 therein). The theoretical results in this section can be seen as a simplification of their result in three dimensions and for a single surface.

As before, we require the surface equation to be square-free and primitive. Additionally, our triangulation method does not apply to surfaces having vertical line components. In this special case, we have to make use of a linear coordinate transformation, such that the surface has no vertical line with respect to the new coordinates. A discussion of this vertical line case can be found in [BKS09].

6.4.1. Triangulation of compact surfaces

Let us start with a compact (thus bounded) surface S, whose silhouette is bounded as well. On the left of Figure 6.5, we see the projected arrangement $\mathcal{A}_S$ for the standard torus, our working example for this paragraph. It contains only two edges and three faces.

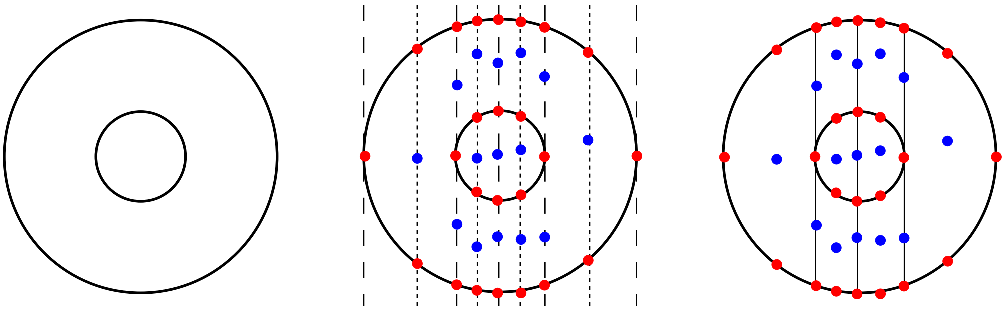


Figure 6.5. The silhouette of the torus.

To triangulate S, we first blow up $\mathcal{A}_S$ to a *cad*.

Definition 6.4.1 (*cad*-stack). Let $\mathcal{A}$ be an arrangement and $x_0 \in \mathbb{R}$. Let $(x_0, y_1), \ldots, (x_0, y_m)$ with $y_1 < \ldots < y_m$ be the set of all points in the fiber of x_0 that are either vertices or lie on an edge of $\mathcal{A}$. Let $q_1, \ldots, q_m - 1$ be rational values such that $y_i < q_i < y_{i+1}$ for $i = 1, \ldots, m-1$. The *cad-stack* at x_0 for $\mathcal{A}$ is the point set

$$\{(x_0, y_1), (x_0, q_1), \ldots, (x_0, y_{m-1}), (x_0, q_{m-1}), (x_0, y_m)\}.$$

The usual definition of a (*cad*-)stack in cylindrical algebraic decomposition also contains rational sample points below y_1 and above y_m. For simplicity, we do not consider them, since the lifts of these points are all empty for a compact surface. Compared to the f-stacks of the silhouette (Definition 3.1.6), the *cad*-stack only contains additional intermediate points between two fiber points. Since we are basically talking about the same object, we simplify the notation and talk about stacks instead of *cad*-stacks.

We perform a curve analysis of the silhouette curve, that is, we create (*cad*-)stacks for the critical x-coordinates of the curve and also for intermediate x-coordinates between two consecutive critical x-coordinates. The result is a collection of critical stacks and intermediate stacks, and it induces an arrangement that refines $\mathcal{A}_S$. We call it $\mathrm{Cad}_S^{(2)}$ (Figure 6.5 (middle)). The lifts of all components of $\mathrm{Cad}_S^{(2)}$ induce another stratification of S, which we call $\mathrm{Cad}_S^{(3)}$.

Lemma 6.4.2. $\mathrm{Cad}_S^{(2)}$ *has* $O(n^6)$ *cells and* $\mathrm{Cad}_S^{(3)}$ *has* $O(n^7)$ *cells.*

Proof. $\mathcal{A}_S$ has up to $O(n^4)$ vertices (Theorem 6.1.9). Each stack contains up to $O(n^2)$ points. This makes $O(n^6)$ vertices at critical positions for $\mathrm{Cad}_S^{(2)}$; the intermediate stacks only double the number of vertices. Since there are up to n lifts over each cell, $\mathrm{Cad}_S^{(3)}$ has $O(n^7)$ cells. □

For a formal definition of the triangulation to be computed, and for the proof that it will indeed be isotopic to S, we next construct a further refined cell decomposition, such that all faces are pseudo-triangles. To *compute* the triangulation, it is not necessary to perform this construction. In our algorithm, only $\mathrm{Cad}_S^{(2)}$ is computed and the triangulation is constructed immediately from it, exploiting the adjacency information in 2D and 3D of the stratification Ω_S.

In each critical stack of $\mathrm{Cad}_S^{(2)}$, we insert vertical straight line edges for consecutive points on the stack. This subdivides the bounded faces into pseudo-polygons. Each such pseudo-polygon contains precisely one non-silhouette point of an intermediate stack in its interior, which we call the *center* of the pseudo-polygon (Figure 6.5 (right)).

Now, for each pseudo-polygon, we insert vertical edges connecting the center v with its lower and upper neighbor on the intermediate stack. Towards all other stack points on the boundary of the pseudo-polygon, we insert x-monotone continuous edges that do not cross each other within the pseudo-polygon. These edges must be lower and upper bounded by the function graphs of the lower and upper boundary segments of the pseudo-polygon. Let φ_1 and $\varphi_2 : [-1,1] \to \mathbb{R}$ denote the corresponding functions. We can assume that both φ_1 and φ_2 meet the intermediate stack of the center at 0. Let y_0 be the y-coordinate of the center. Then, for each each point p on the right boundary of the pseudo-polygon, there exists a parameter $\lambda_p \in [0,1]$ such that $\gamma_p : [0,1] \to \mathbb{R}, t \mapsto y_0 - \varphi_1(0) + \varphi_1(t) + \lambda_p t(\varphi_2(t) - \varphi_1(t))$ connects the center with the boundary point. It is easy to verify that the function graphs are indeed disjoint and bounded by φ_1 and φ_2. For the left boundary, we proceed analogously (Figure 6.6 (left)).

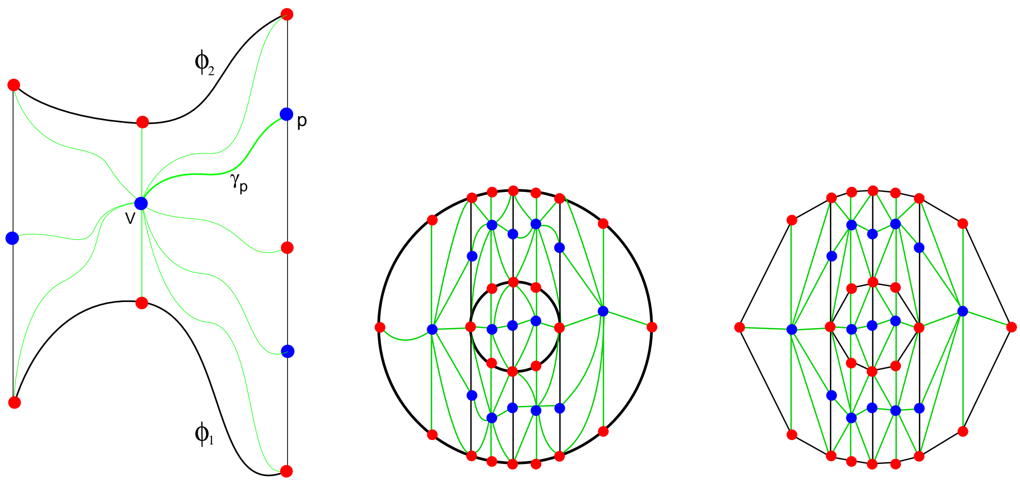


Figure 6.6. The silhouette of the torus (continued).

Because we start with an arrangement $\mathrm{Cad}_S^{(2)}$, consisting of algebraic segments only, the inserted edges are also algebraic. We call the resulting arrangement $\mathrm{PT}_S^{(2)}$ (Figure 6.6 (middle)). The lifts of all cells define another stratification for S, called $\mathrm{PT}_S^{(3)}$.

Lemma 6.4.3. $\mathrm{PT}_S^{(2)}$ *has* $O(n^6)$ *cells and* $\mathrm{PT}_S^{(3)}$ *has* $O(n^7)$ *cells.*

Proof. Comparing the complexities of $\mathrm{Cad}_S^{(2)}$ and $\mathrm{PT}_S^{(2)}$, we note that $\mathrm{PT}_S^{(2)}$ contains additional $O(n^6)$ vertical edges (between consecutive points on the same stack), and up to $O(n^6)$ additional non-vertical edges (each vertex at a critical stack gets up to two additional incident edges). Thus, the complexity of $\mathrm{PT}_S^{(2)}$ equals the complexity of $\mathrm{Cad}_S^{(2)}$, and the same holds for $\mathrm{PT}_S^{(3)}$ and $\mathrm{Cad}_S^{(3)}$. □

Each 2-dimensional cell of $\mathrm{PT}_S^{(2)}$ is a pseudo-triangle, thus, it can be represented by its adjacent three vertices. Note that each pseudo-triangle has at least one adjacent vertex that lies in a face of the original arrangement $\mathcal{A}_S$ (i.e., it does not lie on the silhouette). Finally, we define the triangulation $\mathcal{T}_S$ of S: each patch (i.e., a 2-dimensional cell) of $\mathrm{PT}_S^{(3)}$ is adjacent to three vertices. The triangulation $\mathcal{T}_S^{(3)}$ consists of the union of triangles spanned by these vertex-triples, that is, each "pseudo-triangular" patch is replaced by the actual triangle, defined by the three adjacent vertices.

Theorem 6.4.4. *S is isotopic to $\mathcal{T}_S^{(3)}$.*

Proof. We continuously and bijectively transform S into $\mathcal{T}_S^{(3)}$ in $\mathbb{R}^3$. We proceed in two steps. First, we look at the arrangement $\mathrm{PT}_S^{(2)}$. As mentioned, $\mathrm{PT}_S^{(2)}$ is a pseudo-triangulation of $\mathbb{R}^2$. Let $\mathcal{T}_S^{(2)}$ be the arrangement induced by $\mathrm{PT}_S^{(2)}$ that replaces each curved segment by a straight line (Figure 6.6 (right)). It is not difficult to construct an isotopic map from $\mathbb{R}^2$ to itself that maps vertices, edges, and faces of $\mathrm{PT}_S^{(2)}$ to vertices, edges, and faces of $\mathcal{T}_S^{(2)}$. For that, we define an isotopy H. Fix a point (x_0, y_0) in $\mathbb{R}^2$. If x_0 is a stack coordinate, we set $H((x_0,y_0),t) = (x_0,y_0)$ for each $t \in [0,1]$. If (x_0,y_0) lies on an edge of $\mathrm{PT}_S^{(2)}$, $\mathcal{T}_S^{(2)}$ has a unique (straight) edge with the same two endpoints; let (x_0,y_1) denote the covertical point of that straight edge. Then, we define $H((x_0,y_0),t) := (x_0,(1-t)y_0 + ty_1)$. Finally, if (x_0,y_0) lies on a face of $\mathrm{PT}_S^{(2)}$, consider the edge e_1 that bounds the face from above, and the edge e_2 that bounds the face from below.[57] Let (x_0,y_0^+) denote the covertical point on e_1, and (x_0,y_0^-) denote the one on e_2. Furthermore, let (x_0,y_1^+) denote the covertical point on the edge of $\mathcal{T}_S^{(2)}$ that corresponds to e_1, and let (x_0,y_1^-) denote the covertical point on the edge of $\mathcal{T}_S^{(2)}$ that corresponds to e_2. Define

$$H((x_0,y_0),t) := \left(x_0, (1-t)y_0 + t\left(\frac{y_0 - y_0^-}{y_0^+ - y_0^-}y_1^+ + \frac{y_0^+ - y_0}{y_0^+ - y_0^-}y_1^-\right)\right).$$

Observe that $H(\,\cdot\,,1)$ maps the vertical segment $\{x_0\} \times (y_0^-, y_0^+)$ onto the vertical segment $\{x_0\} \times (y_1^-, y_1^+)$. One can see that H defines a homeomorphism for each t, therefore, it is an isotopy. We remark that the intermediate stacks in our construction are essential for that property, since otherwise, two distinct curved edges can be mapped to the same straight-line edge, thus $H(\,\cdot\,,1)$ would not be bijective.

The transformation can be extended to $\mathbb{R}^3$ by leaving the z-coordinate unchanged. It maps S to an isotopic surface S' and the stratification $\mathrm{PT}_S^{(3)}$ induces a stratification $\mathrm{PT}_{S'}^{(3)}$ of S'. By construction, the cells of $\mathrm{PT}_{S'}^{(3)}$ are lifts of $\mathcal{T}_S^{(2)}$ with respect to S'.

[57] The treatment can be extended to unbounded faces as well, but we will skip the details for brevity

In a second step, S' is transformed into $\mathcal{T}_S^{(3)}$. Again, it is not difficult to see that one can define an isotopy from S' to $\mathcal{T}_S^{(3)}$: A one-dimensional cell of $\mathrm{PT}_{S'}^{(3)}$ that connects two vertices v and w is transformed to the straight edge from v to w, a two-dimensional cell adjacent to u, v, and w is transformed to the triangle spanned by u, v, and w. Note that this transformation only changes the z-coordinates of points, thus, the projection of each cell of $\mathcal{T}_S^{(3)}$ is a cell of $\mathcal{T}_S^{(2)}$. Two distinct cells $C_1, C_2 \in \mathrm{PT}_{S'}^{(3)}$ with $d = \dim C_1 \cap C_2$ are mapped onto cells $C_1^T, C_2^T \in \mathcal{T}_S^{(3)}$ of the same dimension, respectively, such that $d = \dim C_1^T \cap C_2^T$. This is a direct consequence of our decomposition of the plane, which guarantees that each pseudo-triangle contains at least one vertex that is non-critical and each edge connecting two critical points is part of the silhouette curve. Thus two two-dimensional cells (edges) of $\mathrm{PT}_{S'}^{(3)}$ cannot be mapped to the same triangle (or edge, according to the case) of $\mathcal{T}_S^{(3)}$. □

To summarize the subsection, here is a high-level description of our algorithm to compute $\mathcal{T}_S$. We will skip more details for the sake of simplicity.

1. Compute the arrangement $\mathcal{A}_S$ and the stratification Ω_S.
2. Compute the critical and intermediate stacks of $\mathcal{A}_S$ to obtain $\mathrm{Cad}_S^{(2)}$.
3. Compute the list of pseudo-triangles of $\mathrm{PT}_S^{(2)}$. This can be done combinatorially using the adjacency information of $\mathrm{Cad}_S^{(2)}$; an explicit construction of $\mathrm{PT}_S^{(2)}$ is not necessary.
4. Lift each vertex of $\mathrm{Cad}_S^{(2)}$ (see Section 6.2). The lifts define the vertices of $\mathrm{Cad}_S^{(3)}$, which are the same as the vertices of $\mathcal{T}_S$.
5. For each pseudo-triangle of $\mathrm{PT}_S^{(2)}$, consider its lifts. For each lift, compute the three adjacent vertices of $\mathrm{Cad}_S^{(3)}$ using the adjacency information of cells (Section 6.3). Add the triangle spanned by these three vertices to the output list.

Finally, the list of all computed triangles defines the triangulation $\mathcal{T}_S$.

6.4.2. Triangulation of general surfaces

Consider a surface S that is (possibly) unbounded. Clearly, if S is unbounded, it is not possible to produce an isotopic mesh with (finite) triangles. Instead, the triangulation of S is restricted to a (finite) bounding box B that is big enough to contain all "relevant features" of S. By "big enough", we mean that B should contain all bounded cells of Ω_S. Note that, in particular, no vertex of Ω_S is outside the box and, if S is compact, its bounding box contains the whole surface.

The following theorem shows how a bounding box can be computed with algebraic methods. However, since the computation of the boundaries involves quite expensive operations, we subsequently propose an alternative geometric approach for computing a bounding box by an adaptive method.

Theorem 6.4.5. *Let S be a surface with equation f and a projected silhouette curve with equation Γ. For shorter notation, set $f_x := \frac{\partial f}{\partial x}$ and define $f_y, f_z, \Gamma_x, \Gamma_y$ in the same way. Define $C_x := \{x \in \mathbb{R} \mid \exists y \in \mathbb{R} : \Gamma(x,y) = 0 \wedge \Gamma_y(x,y) = 0\}$, $C_y := \{y \in \mathbb{R} \mid \exists x \in \mathbb{R} : \Gamma(x,y) = 0 \wedge \Gamma_x(x,y) = 0\}$, and*

$$\begin{aligned} C_z &:= \{z \in \mathbb{R} \mid \exists x,y \in \mathbb{R} : f(x,y,z) = 0 \wedge \Gamma(x,y) = 0 \wedge (f_x\Gamma_x + f_y\Gamma_y)(x,y,z) = 0\} \cup \\ &\quad\ \{z \in \mathbb{R} \mid \exists x,y \in \mathbb{R} : f(x,y,z) = 0 \wedge f_x(x,y,z) = 0 \wedge f_y(x,y,z) = 0\}. \end{aligned}$$

Let B be a box containing all vertices of Ω_S and all points of the set $C_x \times C_y \times C_z$. Then, B is a bounding box for S.

Proof. Let c be a bounded cell of Ω_S. Clearly, if c is a vertex, it is contained in B, so let it be an edge or a face. Let B' be the projection of B onto the xy-plane, and c' be the projection of c onto the xy-plane. Note that c' is a cell of $\mathcal{A}_S$ by definition of Ω_S.

We show first that c' is inside B'. It is enough to show this for edges, because for faces, we consider the outer boundary cycle, and if each edge of that cycle is inside B', the face must be contained as well. So let c' be an edge. Note that the edge is part of the silhouette curve Γ. Consider a point on the closure of the edge with maximal x-coordinate. Either this point is at the boundary and is thus a vertex, or it is a point in the interior of the edge. In the latter case, it is a local maximum of the silhouette in the x-direction and thus in C_x. In any case, the point with maximal x-coordinate is contained in B. The same argument holds for a point with minimal x-coordinate, and it follows that the whole edge runs inside the x-range of B'. The analogous argument holds for the y-coordinate. Hence, c' is completely inside B'.

It remains for us to show that the z-range of c runs inside the z-range of B. For that, assume first that c is an edge, and let p denote a point on the closure of c with maximal z-coordinate. Either p is at the boundary of c and is thus a vertex of Ω_S, or it is in the interior of c. If p is in the interior of c, consider a C^∞-parameterization $\varphi(t) = (x(t), y(t), z(t))$ of c with $\varphi(t_0) = p$. Since c is on the surface, it is $f \circ \varphi(t) = 0$ for all t, hence $(\nabla f \circ \varphi) \cdot \varphi'(t) = 0$ as well. Since p is a local maximum in z, we have $z'(t_0) = 0$. Furthermore, $x'(t_0) = \Gamma_x(p)$ and $y'(t_0) = \Gamma_y(p)$ holds, thus,

$$(f_x(p), f_y(p), f_z(p)) \begin{pmatrix} \Gamma_x(p) \\ \Gamma_y(p) \\ 0 \end{pmatrix} = 0.$$

In other words, $(f_x\Gamma_x + f_y\Gamma_y)(p) = 0$, thus p is in C_z. For points with minimal z-coordinates, the same argument is valid, so the z-range of c is indeed contained in the z-range of B.

There remains the case of c being a face. Let p be a point on the closure with maximal z-coordinate. If p is at the boundary of c, it is either a vertex or a bounded edge, and since they are contained completely in B, p is also in B. So, let p be in the interior of the face. Since it is a local maximum, its tangent plane is a parallel of the xy-plane. Thus, both f_x and f_y vanish at p, hence, p is in C_z. The same holds for a point with minimal z-coordinate, thus, the z-range of c is contained in B. ☐

We next turn to the alternative iterative approach. Consider the arrangement $\mathcal{A}_S$. Note that for its computation, all edges have been decomposed into x-monotone segments internally, thus, the set C_x (or a superset of it) is already available. We choose a range $r_x := [x_0, x_1]$ containing all points of C_x.

For the y-coordinates, we pick a sample point for each edge of $\mathcal{A}_S$. We choose a range $r_y := [y_0, y_1]$ that contains all y-coordinates of the vertices and the y-coordinates of the sample points. We overlay $\mathcal{A}_S$ with the horizontal lines $y = y_0$ and $y = y_1$. This may cause edges of $\mathcal{A}_S$ to split. Then, each edge of the overlaid arrangement is either completely inside the y-range r_y, or completely outside. If any edge outside r_y is bounded, an insufficiently large interval r_y was chosen, so we enlarge it and retry. An example of such a situation

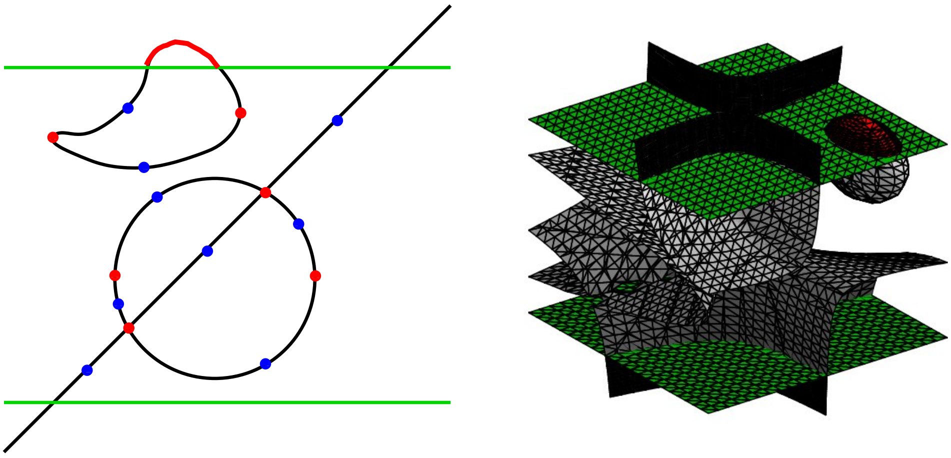


Figure 6.7. Such situations (bounded edges outside the y-range; bounded patches outside the z-range) must be excluded to obtain a correct bounding box.

is depicted on the left of Figure 6.7; observe the red "cap" that leaves the y-range at the top boundary. Otherwise, if all edges outside r_y are unbounded, we are done. Note that it is easy to determine whether an edge is unbounded by checking its two endpoints for finiteness.

For the z-coordinate, we pick sample points for each edge and for each face of $\mathcal{A}_S$. We compute the z-fiber for each vertex and each of those sample points, and choose a range $r_z := [z_0, z_1]$ containing all z-coordinates of the z-fibers. We overlay $\mathcal{A}_S$ with the curves $f(x, y, z_0)$ and $f(x, y, z_1)$. This may cause edges and faces of $\mathcal{A}_S$ to split. We call the overlaid arrangement $\mathcal{A}'_S$; the lifts of its cells induce a stratification Ω'_S of S. Each cell of Ω'_S is completely inside the z-range r_z or completely outside. If any cell of Ω'_S is outside r_z and bounded, an insufficiently large r_z was chosen, so we enlarge it and retry. On the right of Figure 6.7, there is a situation where this happens; observe the red "cap" that leaves the z-range at the top boundary. Otherwise, if all cells outside r_z are bounded, we are done. Note that, again, it easy to determine whether a cell of Ω'_S is bounded by checking its adjacent vertices for finiteness.

By construction, the box $B := [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1]$ is a bounding box. We now explain how to triangulate S inside that box. Overlay $\mathcal{A}_S$ with the vertical lines $x = x_0$, $x = x_1$, with the horizontal lines $y = y_0$, $y = y_1$, and the curves $f(x, y, z_0)$ and $f(x, y, z_1)$. Throw away any component that is outside the box $[x_0, x_1] \times [y_0, y_1]$ (Figure 6.8 (left)). Create a stack at each x-critical coordinate of the overlaid arrangement, that is, at each x-coordinate of a vertex. Also, create an intermediate stack between two critical stacks. This results in an arrangement $\mathrm{Cad}'^{(2)}_S$.

From this point onwards, we proceed as in the case of compact surfaces. The arrangement $\mathrm{Cad}'^{(2)}_S$ is decomposed into pseudo-triangles (as depicted on the right of Figure 6.8). Each pseudo-triangle has several lifts on the surface. Note that a lifted pseudo-triangle is either completely inside B or completely above B, or completely below B. If it is inside B, we add the triangle defined by the three adjacent vertices to $\mathcal{T}_S$, otherwise, we ignore the triangle. After doing so for each pseudo-triangle, the set of triangles $\mathcal{T}_S$ triangulates S inside B.

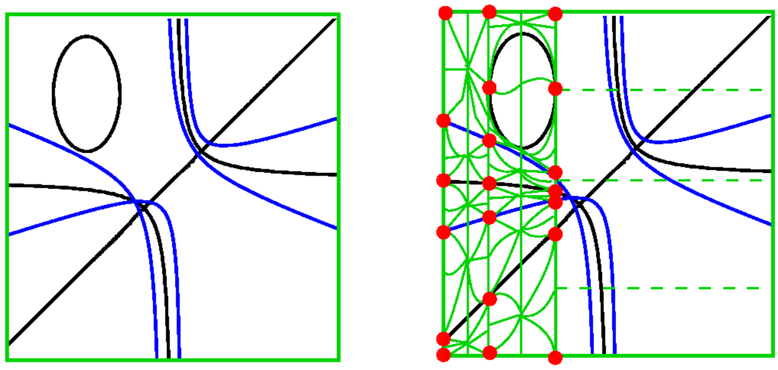


Figure 6.8. The projection phase in the unbounded case.

Summary

By extending the stratification into a *cad* of the surface (with adjacency information), an isotopic triangulation of the surface can be computed. The amount of additional computations is quite low, but the complexity of the decomposition increases from $O(n^5)$ to $O(n^7)$. Unbounded surfaces can be triangulated within a bounding box that is computed adaptively.

6.5. Implementation and experiments

6.5.1. Stratification

Our stratification algorithm is transformed into a fully working C++-implementation based on CGAL. Algebraic surfaces are represented by the class template `Algebraic_surface_3`. To construct and refine the n-k-arrangement for a surface S, we rely on the software framework presented in Section 4.2 of this thesis. We have seen that, for technical reasons, curves are split into x-monotone subcurves. Our traversal combines them to maximal n-k-constant paths. We make extensive use of advanced operations on arrangements provided by the corresponding CGAL package. For example, we attach a collection of information (e.g., n_C and k_C) to each DCELcomponent. In combination with CGAL's overlay mechanism, the computation of $\mathcal{A}_S$ can be implemented as explained in Section 6.1. Additionally, the construction of z-fibers, as presented in Section 6.2, benefits from the precomputed parameters n_C and k_C for each cell. This avoids repeating costly tests; for example, whether a point lies on some curve. We also follow the scheme of lazy-evaluation (compare page 160), for example, the sample point for a cell and its z-fiber is only computed on demand and then cached.

We briefly mention that our design of implementation decouples combinatorial and generic tasks from surface-specific ones using the generic programming paradigm. In particular, three operations that follow our algorithmic description are expected from a supported surface. First, the decomposition of the polynomials $\mathrm{res}_z(f, \frac{\partial f}{\partial z})$, a_i, and $\mathrm{stha}_i(f_n)$ into square-free factors and construct corresponding curve instances. Second, the construction of a z-fiber for given p with respect to a surface, knowing n_p and k_p. Third,

Instance	$\deg_{x,y,z}$	$\lvert\Omega_S\rvert$	(V,E,F)	**total**	proj	stacks	adj	unfiltered
whitney-umbrella	(2,1,2)	9	(1,4,4)	**0.02**	16%	33%	33%	0.02
cayley-cubic	(2,2,2)	22	(3,10,8)	**0.45**	42%	23%	29%	0.50
steiner-roman-surface	(2,2,2)	27	(5,12,8)	**0.24**	11%	31%	53%	0.26
comlumpius	(3,3,3)	38	(5,11,7)	**0.88**	75%	8%	6%	1.34
bohemian-dome	(4,4,4)	61	(7,20,14)	**1.34**	16%	9%	72%	1.85
chair	(4,4,4)	31	(4,9,7)	**2.23**	84%	3%	10%	3.59
chub-surface	(4,4,4)	52	(8,8,6)	**0.45**	23%	7%	64%	0.41
tangle-cube	(4,4,4)	28	(0,6,7)	**0.44**	60%	12%	23%	0.78
tetrahedral-skeleton	(4,4,4)	26	(5,12,8)	**0.75**	21%	12%	64%	0.82
dupin-cyclide	(4,4,4)	10	(3,4,4)	**0.15**	60%	26%	7%	0.26
seahorse	(4,6,3)	14	(2,6,5)	**4.55**	1%	89%	8%	4.52
star	(6,6,6)	5	(1,1,2)	**5.04**	98%	0%	0%	20.00
sweet	(6,6,6)	9	(1,2,3)	**1.17**	89%	9%	0%	2.81
hunt	(6,6,6)	15	(3,2,3)	**1.30**	84%	9%	6%	–
spiky	(6,9,6)	13	(1,8,8)	**0.50**	39%	17%	42%	0.62
zipf	(6,6,6)	5	(1,1,2)	**0.32**	93%	4%	1%	0.28
C8	(8,8,8)	496	(40,48,26)	**25.02**	62%	7%	28%	22.18
rand-3	(3,3,3)	15	(2,3,3)	**0.12**	43%	16%	10%	0.22
rand-4	(4,4,4)	64	(7,14,8)	**3.40**	82%	4%	8%	5.20
rand-5	(5,5,5)	154	(16,24,10)	**140.64**	97%	1%	0%	345.49
interpolated-3	(3,3,3)	23	(4,6,3)	**0.24**	65%	11%	8%	0.46
interpolated-4	(4,4,4)	66	(10,16,7)	**24.24**	96%	1%	1%	48.27
projected-4d	(4,4,4)	34	(4,12,9)	**8.96**	95%	1%	2%	14.96

Table 6.1. Complexity and running times (in seconds) of the stratification algorithm for a selection of surfaces.

for two adjacent cells of $\mathcal{A}_S$, the computation of their lifted adjacencies (see Section 6.3 for details). We plan to augment our implementation for multiple surfaces. A generic framework for this task has already been established in [BS08].

Experiments:

We run experiments on our implementation using well-known examples from algebraic geometry.[58]. These surfaces cover many possible geometric features of surfaces; for instance, Steiner-roman and Cayley-cubic contain a vertical line component. We also constructed surfaces with random coefficients and surfaces of degree 3 and 4 by interpolation of randomly chosen sample points. A final example is a projection of the intersection of two random quadrics in 4D into the three-dimensional space. All experiments are executed on the same benchmark machine as all previous experiments.

Table 6.1 states the size of the n-k-arrangement $\mathcal{A}_S$, the total number of cells in Ω_S, and the running times obtained for a selection of tested surfaces. It is expected that (some) surfaces do not show any n-k-vertex (e.g., `tangle-cube`), or n-k-edge (e.g., xy-functional surfaces) at all. We also list the relative timings to compute the n-k-arrangement (proj), to compute z-stacks at sample points (stacks), and to compute the adjacencies between cells (adj). Our implementation allows switching off the m-k-bitstream-Descartes method in the z-stack computation step (in this case, the square-free part is computed for each

[58] Subsets of the tested example surfaces are provided courtesy by the AIM@SHAPE Shape Repository of INRIA, by `www.singsurf.org`, by `www.freigeist.cc`, and by [PV07]

non-square-free polynomial). The total running time for this is also given (unfiltered).

We observe that the construction of $\mathcal{A}_S$ is the bottleneck in many examples. In particular, when considering randomly generated surfaces, this computation limits the usability for higher degrees. This is no surprise, since we have to analyze plane algebraic curves of degrees of up to $n(n-1)$. To prove the benefit of our approximative and combinatorial methods, we compare the total running time with the unfiltered approach. Often, the lifting and the adjacency computation become significantly slower without that filtering step (star, rand-5) – for the hunt surface, the unfiltered version did not terminate even after several hours.[59] Some surfaces are even faster with the unfiltered version (C8, chub). This is simply because they are in a degenerate situation where the m-k filter fails in most situations.

6.5.2. Triangulation

We have a preliminary implementation to compute a triangulation of a surfaces within a bounding box.[60] This bounding box must be defined by the user – our ideas from Section 6.4.2 to obtain a box adaptively have not been implementated yet.

The current implementation supports the refinement of the triangulation of $\mathcal{A}_S$, and in that way, of the lifted triangulation of the surface (controlled by a parameter). This makes it possible to triangulate the surface into few triangles (to represent its topology) as well as to compaute a close geometric approximation to get topologically reliable and accurate plots (Figure 6.9). The output of the algorithm is simply a collection of index triples, where each index represents a point in space. We admit that some special cases (for instance, 1-dimensional components of a surface) are not yet covered, so we do not call our implementation of the triangulation complete in its current state.

Regarding the performance, we observed that when we compute an isotopic triangulation with as few triangles as possible, the costs are dominated by the costs of the stratification. Indeed, almost all information of the *cad* in the projection plane is already computed during the stratification, it only remains to compute the intermediate points on the stacks and lift them; these steps are negligible, at least if they are not performed too often. If the number of triangles increases, however, the high number of stacks and intermediate steps increasingly influences the practical performance. We remark that the computation of many lifts would be a perfect scenario for parallel computations, as stacks can be computed independently from each other.

Summary

We provide the first complete EGC-implementation to compute a stratification of an algebraic surface and we have extended the implementation to compute an isotopic triangulation. Because there is no comparable implementation available that returns a certified output, we have shown that the stratification (and triangulation) of several reference surfaces from algebraic geometry, including all sorts of degeneracies, can be computed quite fast. As for the case of algebraic curves, a comparison with a certified method that is based on subdividing the space would be interesting for the future.

[59] We admit that this might also be caused by a bug in the code.

[60] The code is currently maintained and extended by Pavel Emeliyanenko and is based on an earlier version by the author of this thesis.

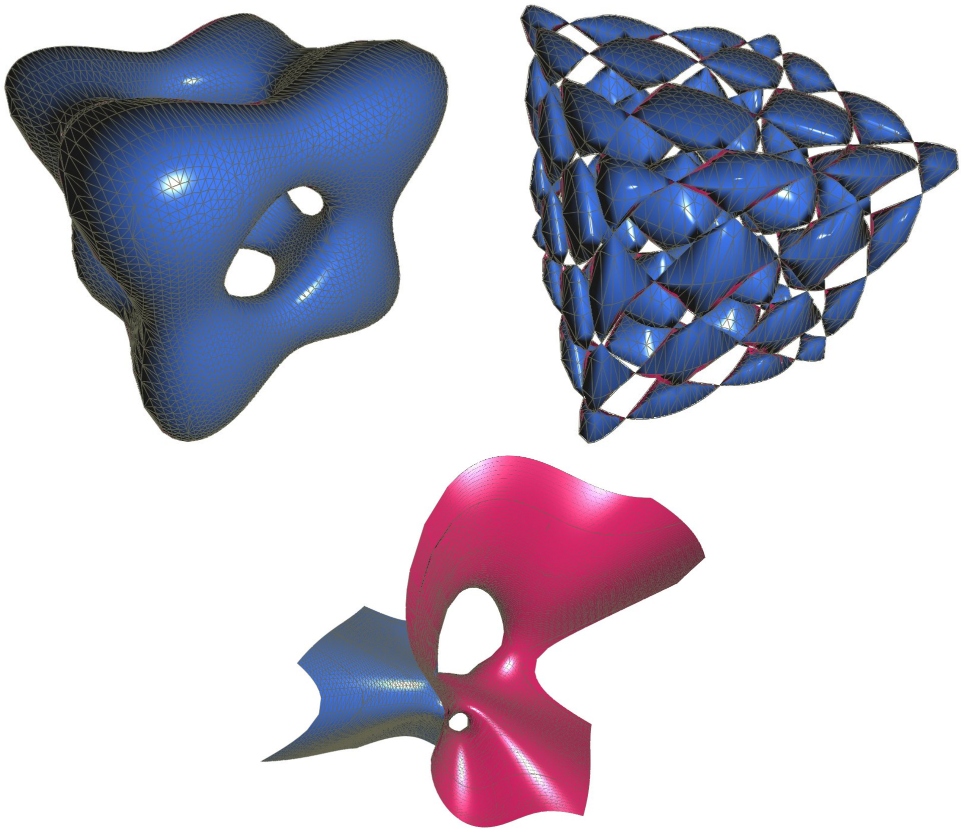


Figure 6.9. Triangulations of the tangle cube, the C8 surface, and the columpius surface.

6.6. Bounds for the size of isocomplexes

As the final subject of this thesis, we turn to the following question (more precisely, four questions): *How many vertices are needed to define a stable/general isocomplex for an algebraic curve/surface of degree n?* This thesis has described algorithms for stable isocomplexes for curves and surfaces. Indeed, the curve analysis implicitly defines a stable isocomplex: one that has an f-stack at all critical x-coordinates, intermediate values in between, and segments between stacks that can be isotopically transformed to straight-line segments. At most $O(n^3)$ vertices are produced by the curve analyis algorithm. We will show that this bound is tight, which means that there are curves of degree n such that a stable isocomplex must consist of $\Omega(n^3)$ vertices. This is somewhat surprising, because the stack-based approach produces a lot of unnecessary vertices due to coverticalness to a critical point. If we relax the requirement of being stable, we can give a simple algorithm that produces an isocomplex with $O(n^2)$ vertices only, which is also tight,

For surfaces, our results give differing lower and upper bounds: In this chapter, we have derived an $O(n^7)$ stable triangulation, which is the best upper bound we are aware of. We construct a family of surfaces for which we can prove a lower bound of $\Omega(n^4)$ vertices for any stable isocomplex. In the general case, we can prove a lower bound of

$\Omega(n^3)$ and an upper bound of $O(n^5)$. An extended abstract of the results in this section appears in [KS09].

6.6.1. General isocomplexes

We first look at general (or not-necessarily stable) isocomplexes, that is, ones where we are allowed to move vertices under the isotopy. Let us look first at the planar case. So, let $C_f := V(f)$ be a curve of degree n. We know already that C_f can have up to $O(n^2)$ many critical points, but not every critical point has to be represented by a vertex in the isocomplex: the neighborhood of regular x-extreme points and even singular points with two adjacent segments can be isotopically transformed into a straight-line. However, if a singular point is isolated, or has more than two adjacent segments, its image in the isocomplex must be a vertex.

For the lower bound of $\Omega(n^2)$ vertices, it is enough to construct a curve with many self-intersections. The simplest approach is to choose n lines in the plane in generic position, which means that no two of them are parallel, and no three of them meet at a common point. The union of these lines defines a curve of degree n, and the number of self-intersections is obviously $\binom{n}{2} = \Theta(n^2)$. Thus, any isocomplex for this curve requires at least $\Omega(n^2)$ vertices.

For surfaces, a lower bound of $\Omega(n^3)$ vertices is obtained in a similar way. Consider n planes in generic position: each triple meets at a unique point, and this point must appear as a vertex in the isocomplex, because its local topology is different from any neighboring point. There are $\binom{n}{3}$ such points, and consequently, at least $\Omega(n^3)$ vertices in the isocomplex.

We turn to the upper bounds. In the planar case, we show how the output of the curve analysis (or equivalently, of the *cad* of one curve) can be simplified such that the resulting graph is of complexity $O(n^2)$. The output of the curve analysis yields $O(n^3)$ points on the fibers of critical x-coordinates and at intermediate positions in between. Recall the definition of an *event point* (Definition 3.2.12) as a point whose branch numbers are not $(1, 1)$. For any non-event point at a fiber, the *predecessor* is the (unique) point on the left-neighboring fiber that is connected to it. A non-event point is called a *transit point*, if its predecessor is a non-event point as well. The simple idea is to arrange sequences of consecutive transit points on the same horizontal line while keeping the vertical ordering of the fiber points of one x-coordinate intact.

In detail, the algorithm can be formulated as follows. Iterate through the stacks (critical and intermediate) from the left to right. On the first stack, set the y-coordinate of the i-th fiber point (counted from below) to i, such that the stack points have y-coordinates $0, 1, \ldots, m$ (this makes sure that all vertices will have rational coordinates in the end). For the i-th stack with $i > 1$, consider the transit points first. Let p be such a transit point, and $\mathrm{pre}(p)$ be its predecessor in the $(i-1)$-th stack. Set the y-coordinate of p to the same value as the y-coordinate of $\mathrm{pre}(p)$ (the segment connecting them is horizontal). Insert the non-transit points such that the final stack has the correct vertical ordering (see Figure 6.10 for an illustration of the algorithm). In a final step, remove all vertices that have only two adjacent horizontal edges, and merge the two edges into one.

Why does the described algorithm produce $O(n^2)$ vertices? Observe that the resulting graph has vertices only at event points, and at stack points that are adjacent to event points. Since event points are critical, there are at most n^2 of them. It suffices to count

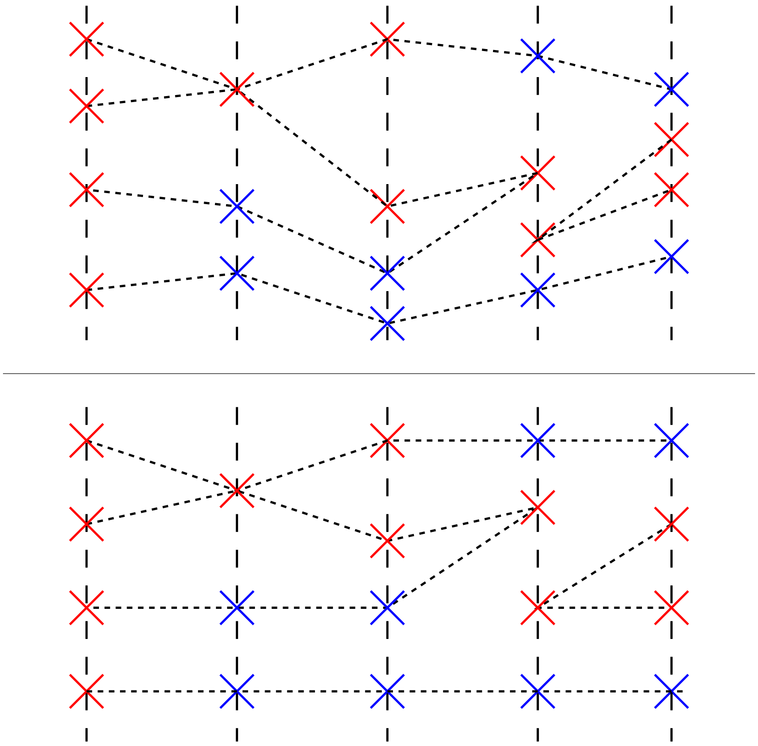


Figure 6.10. The top picture shows the original stacks (transit points in blue); the bottom picture shows the simplified graph.

the number of fiber points connected to event points. This is the same as counting the number of incident branches for each event point.

Lemma 6.6.1. *For an event point p, we set b_p to the sum of its branch numbers. The sum of the b_p's for all event points is bounded by $4n^2$.*

Proof. It is enough to argue that the intersection multiplicity (Definition 2.3.32) of $V(r)$ and $V(\frac{\partial r}{\partial y})$ at an event point p with $b_p = 2k_p$ is at least $k_p - 1$: Because the sum of the intersection multiplicities is bounded by n^2, it then follows with P the set of all event points that

$$\sum_{p \in P} b_p = 2 \sum_{p \in P} (k_p - 1) + 2\#P \leq 4n^2.$$

Consider a singular point p with $b_p = 2k$. W. l. o. g., we can assume that p is the origin, that none of the branches is vertical, and that there is no covertical singularity (otherwise, we can translate and rotate the curves, which leaves the branch numbers and the intersection multiplicities the same). It is not difficult to see that 0 is a root of $f(0, y)$ of multiplicity at least k: There are at least k branches of the curve approaching the origin from the same side, say from the right. Then, $f(\varepsilon, y)$ has k roots that converge to 0 when ε approaches 0, and by continuity reasons, 0 is a k-fold root of $f(0, y)$. For that reason, 0 is also a $(k-1)$-fold root of $f'(0, y)$.

By that argument, we can write f and $\frac{\partial f}{\partial y}$ in the following way:

$$f(x,y) = xf_1(x,y) + y^k f_2(x,y), \quad \frac{\partial f}{\partial y}(x,y) = xf_3(x,y) + y^{k-1} f_4(x,y)$$

Considering the Sylvester matrix of f and $\frac{\partial f}{\partial y}$, we observe that any non-zero entry in its last $(k-1)$ columns contains a factor x. Hence, we can factor out a factor of x in each of these columns, which yields a factor of x^{k-1} in the resultant.

□

It follows that we produce indeed $O(n^2)$ many vertices in the algorithm. We turn to the surface case, and build an $O(n^5)$ triangulation. We start as we did for the triangulation algorithm described in this chapter, that is, we construct the silhouette and perform a curve analysis on it, which yields $O(n^6)$ many vertices. Next, we apply the simplification on the silhouette. This yields an isotopic straight-line graph with a reduced complexity of $O(n^4)$. Let H denote the isotopy between them. We compute a *trapezoidal map* of the graph [dBvKOS00], that is, we draw a vertical line from each vertex to the next upper and lower neighbors. Then, for each trapezoid obtained (or triangle, in a degenerate case), we pick a point in its interior and connect it to any vertex at the boundary (there are at most 4 such vertices). These steps clearly do not increase the asymptotic complexity of the graph and we obtain a triangulation of the silhouette with $O(n^4)$ vertices. This triangulation can be lifted to a triangulation of the surface just by exploiting the adjacency information from the corresponding z-stacks. Since there are up to n lifts per vertex, we obtain a triangulation with up to $O(n^5)$ vertices.

We were not able to close the gap between the lower and upper bounds in the surface case. The natural idea of proceeding in analogy to the simplification in $\mathbb{R}^2$, that is, by performing a plane sweep in $\mathbb{R}^3$, and connecting "transit edges" by horizontal surface patches seems not to work out. The problem is that two patches can change their vertical ordering without intersecting each other. For curves, this is impossible, and this is a crucial property for obtaining an $O(n^2)$ isocomplex. Still, we believe that our upper bound of $O(n^5)$ triangles is not optimal. A more economical triangulation should be achievable by a method that is not based on projection and lifting.

6.6.2. Stable isocomplexes

For stable isocomplexes, the upper bounds of $O(n^3)$ (for curves) and $O(n^7)$ (for surfaces) follow from the results of this thesis (and also from *cad* algorithms). We seek lower bounds.

We first describe the construction idea for curves (the idea for surfaces is similar). Assume that the unit circle is part of the curve. A stable isocomplex for the curve has to contain a cycle with points on the unit circle. If additionally, isolated points inside the unit circle and close to its boundary belong to the curve, this cycle is forced to also include those points. Indeed, we can define a set of $\Theta(n^2)$ isolated points such that the cycle of the unit circle consists of $\Omega(n^2)$ vertices, see Figure 6.11 for an illustration. Performing the same construction not just for the unit circle, but for $\Theta(n)$ concentric circles, all with radius close to one, yields $\Omega(n^3)$ vertices for the isocomplex.

The main question is how to force isolated points at fixed positions to be part of the curve (or the surface). The next theorem states that one can freely pick $\Theta(n^2)$ arbitrary small regions in $\mathbb{R}^2$ ($\Theta(n^3)$ in $\mathbb{R}^3$) and always finds a curve (or surface) of degree n that contains an isolated point in any of those regions.

Theorem 6.6.2. *For $d, n \in \mathbb{N}$, set $c := \binom{\lfloor n/2 \rfloor + d}{d} - d$. Then, for any $\varepsilon > 0$, and any set of points $p_1, \ldots, p_c \in \mathbb{Q}^d$, there exists a hypersurface $C \subset \mathbb{R}^d$ of degree n such that for any p_i, C contains an isolated point $p'_i \in \mathbb{R}^d$ with $\|p_i - p'_i\|_2 < \varepsilon$.*

Proof. W. l. o. g., we assume that n is even. The idea is to construct d polynomials $f_1, \ldots, f_d$ of degree $n/2$ that all interpolate the points $p_1, \ldots, p_c$, and to consider the curve defined by $f := f_1^2 + \ldots + f_d^2$. Obviously, $\deg f \leq n$, and $V(f)$ has isolated points exactly at the intersection points of $V(f_1) \cap \ldots \cap V(f_d)$. We have to prove that $f_1, \ldots, f_d$ can be chosen such that they intersect only in a finite number of points.

First of all, almost all choices of d hypersurfaces of degree $\frac{n}{2}$ yield a zero-dimensional common intersection: consider the coefficients of the polynomials as indeterminates, then the (multivariate) resultant R [CLO05] with respect to any variable, say x_1, is a polynomial in x_1 that does not vanish completely. This means that for almost any choice of coefficients, the concrete set of polynomials will only have finitely many common intersections.

We next fix c points $p'_1, \ldots, p'_c$ in $\mathbb{C}^d$ with yet indeterminate coordinates. We force d hypersurfaces, with indeterminate coefficients to pass through them. As a consequence, each coefficient can be re-expressed in dependency of the coordinates of the p'_i, plus additional degrees of freedom. The same also holds true for the resultant polynomial R. The statement of the theorem follows, if we can prove that the resultant polynomial does not vanish identically for all choices of $p'_1, \ldots, p'_c$, because this already implies that it does not vanish identically for almost all choices of $p'_1, \ldots, p'_c$.

The degree of R is $(n/2)^d$. Choose d hypersurfaces $f_1, \ldots, f_d$, such that the leading term of R does not vanish. Then, there exist $(n/2)^d$ intersection points in the projective space $\mathbb{P}(\mathbb{C}^d)$, and we can w. l. o. g. assume that all these points actually lie in the affine space $\mathbb{C}^d$. It is a simple proof that $(n/2)^d \geq c$ for all $n, d \in \mathbb{N}$ (by induction on d). So, we can pick c of the common intersection points as points $p'_1, \ldots, p'_c$ from above, and set the other degrees of freedom such that we obtain $f_1, \ldots, f_d$. With this choice, the resultant does not vanish, thus, it defines a lower-dimensional vaariety in $\mathbb{C}^d$. It follows that the resultant does not vanish for almost any choice of base points $p'_1, \ldots, p'_c$.

Thus, for given points $p_1, \ldots, p_c \in \mathbb{Q}^d$, we find points $p'_1, \ldots, p'_c$ in an ε-ball around them such that there are hypersurfaces $f_1, \ldots, f_d$ interpolating them and such that the resultant of $f_1, \ldots, f_d$ does not vanish completely. It remains to argue that $p'_1, \ldots, p'_c$ can be chosen with real coordinates, but this follows immediately, since otherwise, the resultant variety would contain an open ball of $\mathbb{R}^d$, and consequently, it would contain the whole $\mathbb{R}^d$, which is impossible. □

Let us apply this theorem in the plane. Consider the unit circle, called s. The isocomplex for s is determined by a sequence of points on the circle. We cut out $c' := \binom{n/4+2}{2} - 2$ disjoint regions of the unit disc, by intersecting the disc with lines. We place a disc of size ε in each of the regions and force an isolated point of the curve to lie inside each disc (Figure 6.11). By the above theorem, this is possible using an algebraic curve of degree $n/2$.

We can observe that the isotpoic cycle of the unit circle contains a vertex in each of the regions. Obviously, if there is no such vertex, the cycle misses the region completely, so the isolated point is outside the cycle, but that contradicts the properties of a stable isocomplex. This means that at least $c' = \Omega(n^2)$ vertices are placed on the unit circle.

Finally, we take a collection of $n/4$ concentric circles, instead of the unit circle, such

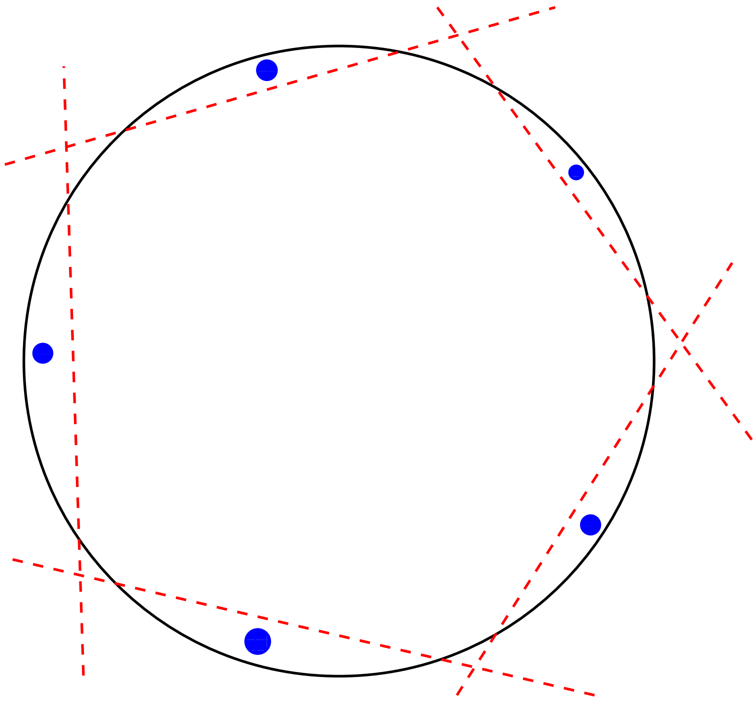


Figure 6.11. To contain points in the blue discs, a vertex must be placed in each of the regions cut out by the dashed lines.

that the lines chosen as in Figure 6.11 still cut out c' disjoint regions for any of the circles. This is clearly possible, if all concentric circles have radius close enough to 1. The argument from above now works separately for each of the circles, thus, each one is divided into $\Omega(n^2)$ line segments under the isotopy.

To summarize, the final curve consists of two components: one curve of degree $n/2$ that forces the isolated singularities in the regions, and a collection of $n/4$ circles (of degree $n/2$). The union is of degree n, and any stable isocomplex requires $\Omega(n^3)$ line segments (and vertices) in total.

The reader might object to the fact that the constructed curve is reducible, but irreducibility can be achieved as well by slightly changing the construction. Construct two distinct (irreducible) curves that contain the same set of isolated points; let g_1 and g_2 denote their defining polynomials. Now, consider $n/2$ circles with radius close to one, just as before. Let $C_1, \ldots, C_{n/2}$ denote them, sorted by increasing radius. Let $V(f_1)$ be the curve defined by the union of the circles with odd index, and $V(f_2)$ the curve defined by the even circles. In other words, any ray from the origin meets the circles of f_1 and f_2 in an alternating sequence.

Define $f := f_1 g_1 + f_2 g_2$. In general, this is an irreducible curve. We have to argue why this curve still yields $\Omega(n^3)$ line segments in a stable isocomplex. For that, consider a ray starting in the origin. This ray intersects each circle exactly once. Let $s_1, \ldots, s_{n/2}$ be the signs of f at the intersection points. It is not difficult to see that this sign sequence is $s_i = 1$ if $i = 2, 3 \mod 4$, and $s_1 = -1$ otherwise (Figure 6.12). Thus, f must have a circular component in the annulus between C_i and C_{i+1} for $i = 1, 3 \mod 4$, and the isocomplexes of these $n/4$ circular components each have to introduce at least $\Omega(n^2)$ vertices, by the same argument as for the concentric circles.

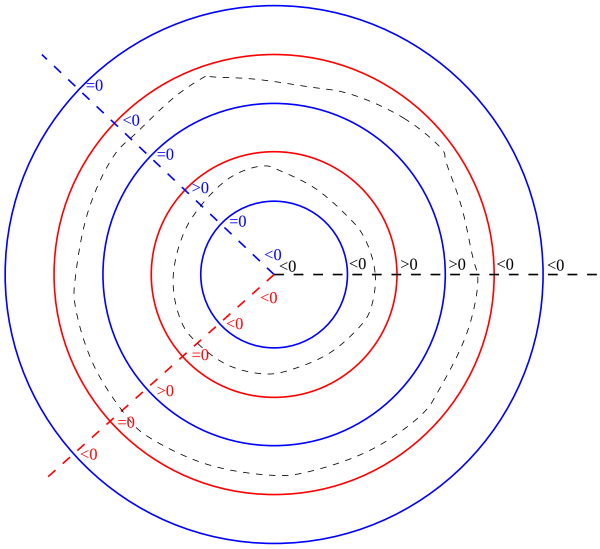


Figure 6.12. Illustration of an irreducible worst-case construction. Let $V(f_1)$ be defined by the blue circles, and $V(f_2)$ by the red circles. For any ray from the origin, f_1 changes its sign whenever passing a point of $V(f_1)$, and the same holds for f_2 (denoted by the blue and red rays on the right-hand side). Since $f = f_1g_1 + f_2g_2$, and because $g_1, g_2 \geq 0$, the sign of f at an intersection with a circle is determined by the sum of f_1 and f_2. Therefore, we must have circular components of f in every second annulus.

The construction for surfaces is completely analogous. We consider the unit sphere and choose $c' := \binom{n/4+3}{3} - 3 = \Theta(n^3)$ disjoint regions by intersecting the unit ball with planes. Inside each region, we place a small ball, and force an isolated point in the ball; these isolated points can be chosen to lie on an algebraic surface of degree $n/2$. By the same argument as in $2D$, a vertex on the unit sphere must be placed in each region. This yields $\Omega(n^3)$ regions for the sphere and, choosing $n/4$ concentric spheres, we obtain at least $\Omega(n^4)$ vertices in total.

Summary

We have analyzed the complexity of (stable) isocomplexes for algebraic curves and surfaces. For algebraic curves, we were able to give tight bounds: $\Theta(n^2)$ simplices are necessary and sufficient for general isocomplexes (in the worst case); for the stable case, $\Theta(n^3)$ is tight. For surfaces, we only know that $\Omega(n^3)$ simplices are necessary and $O(n^5)$ are sufficient in the general case. In the stable case, the bounds are $\Omega(n^4)$ and $O(n^7)$.

Bibliography

[Abb06] John Abbott. Quadratic Interval Refinement for Real Roots. Poster presented at the 2006 International Symposium on Symbolic and Algebraic Computation (ISSAC 2006), 2006.

[ACM84a] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical Algebraic Decomposition I: The Basic Algorithm. *SIAM Journal on Computing*, 13:865–877, 1984. Reprinted in [CJ98], pages 136–151.

[ACM84b] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical Algebraic Decomposition II: An Adjacency Algorithm for the Plane. *SIAM Journal on Computing*, 13:878–889, 1984. Reprinted in [CJ98], pages 152–165.

[ACM88] Dennis S. Arnon, George E. Collins, and Scott McCallum. An Adjacency Algorithm for Cylindrical Algebraic Decompositions of Three-Dimensional Space. *Journal of Symbolic Computation*, 5:163–187, 1988.

[ADTGV04] Jounaidi Abdeljaoued, Gema M. Diaz-Toca, and Laureano Gonzalez-Vega. Minors of Bezout Matrices, Subresultants and the Parameterization of the Degree of the Polynomial Greatest Common Divisor. *International Journal of Computer Mathematics*, 81(10):1223–1238, 2004.

[AH83] Gotz Alefeld and Jurgen Herzberger. *Introduction to Interval Computations*. Computer Science and Applied Mathematics. New York Academic Press Inc., 1983.

[Akr] Alkiviadis G. Akritas. A new Look at One of the Bisection Methods Derived from Vincent's Theorem or There is no Descartes' Method. Available at `http://inf-server.inf.uth.gr/~akritas/articles/71.pdf`.

[AM88] Dennis Arnon and Scott McCallum. A Polynomial Time Algorithm for the Topological Type of a Real Algebraic Curve. *Journal of Symbolic Computation*, 5:213–236, 1988.

[AMT09] Lionel Alberti, Bernard Mourrain, and Jean-Pierre Técourt. Isotopic Triangulation of a Real Algebraic Surface. *Journal of Symbolic Computation*, 44:1291–1310, 2009.

[AMW08] Lionel Alberti, Bernard Mourrain, and Julien Wintz. Topology and Arrangement Computation of Semi-algebraic Planar Curves. *Computer Aided Geometric Design*, 25:631–651, 2008.

[Apo74] Tom M. Apostol. *Mathematical Analysis*. Addison-Wesley, 2nd edition, 1974.

[Arn88] Dennis S. Arnon. A Cluster-Based Cylindrical Algebraic Decomposition Algorithm. *Journal of Symbolic Computation*, 5:189–212, 1988.

[AS00] Pankaj K. Agarwal and Micha Sharir. Arrangements and Their Applications. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of computational geometry*, pages 49–119. North-Holland Publishing Company, 2000.

[AS05] Juan G. Alcázar and Juan R. Sendra. Computation of the Topology of Real Algebraic Space Curves. *Journal of Symbolic Computation*, 39:719–744, 2005.

[ASS07] Juan G. Alcázar, Josef Schicho, and Juan R. Sendra. A Delineability-based Method for Computing Critical Sets of Algebraic Surfaces. *Journal of Symbolic Computation*, 42:678–691, 2007.

[BCGY08] Michael Burr, Sung W. Choi, Benjamin Galehouse, and Chee K. Yap. Complete Subdivision Algorithms, II: Isotopic Meshing of Singular Algebraic Curves. In *Proceedings of the 2008 International Symposium on Symbolic and Algebraic Computation (ISSAC 2008)*, pages 87–94. ACM press, 2008.

[BCL82] Bruno Buchberger, George E. Collins, and Rüdiger Loos, editors. *Computer Algebra – Symbolic and Algebraic Computation*. Springer, 1982.

[BCSM+06] Jean-Daniel Boissonnat, David Cohen-Steiner, Bernard Mourrain, Günter Rote, and Gert Vegter. Meshing of Surfaces. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 181–229. Springer, 2006.

[BCW07] Eric Berberich, Manuel Caroli, and Nicola Wolpert. Exact Computation of Arrangements of Rotated Conics. In *Proceedings of 23rd European Workshop on Computational Geometry (EWCG 2007)*, pages 231–234. Technichal University of Graz, Austria, 2007.

[BE08] Eric Berberich and Pavel Emeliyanenko. CGAL's Curved Kernel via Analysis. Technical Report ACS-TR-123203-04, Max-Planck-Institute for Informatics, Saarbrücken, Germany, 2008.

[BEH+02] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Kurt Mehlhorn, and Elmar Schömer. A Computational Basis for Conic Arcs and Boolean Operations on Conic Polygons. In *Proceedings of 10th European Symposium on Algorithms (ESA 2002)*, volume 2461 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2002.

[BEH+05] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Lutz Kettner, Kurt Mehlhorn, Joachim Reichel, Susanne Schmitt, Elmar Schömer, and Nicola Wolpert. EXACUS: Efficient and Exact Algorithms for Curves and Surfaces. In Gerth S. Brodal and Stefano Leonardi, editors, *Proceedings of 13th Annual European Symposium on Algorithms (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 155–166, 2005.

[BEPP97] Hervé Brönnimann, Ioannis Z. Emiris, Victor Y. Pan, and Sylvain Pion. Computing Exact Geometric Predicates Using Modular Arithmetic with Single Precision. In *Proceedings of 13th Annual Symposium on Computational Geometry (SoCG 1997)*, pages 174–182. ACM press, 1997.

[Ber08] Eric Berberich. *Robust and Efficient Software for Problems in 2.5-Dimensional Non-Linear Geometry*. PhD thesis, Saarland University, Saarbrücken, Germany, 2008.

[Bez07] Helmut E. Bez. Rational Maximal Parametrisations of Dupin Cyclides. In Ralph R. Martin, Malcolm A. Sabin, and Joab R. Winkler, editors, *Mathematics of Surfaces XII*, volume 4647 of *LNCS*, pages 78–92. Springer, 2007.

[BFG+08] Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Stefan Schirra, and Sylvain Pion. 2D and 3D Geometry Kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

[BFH+07] Eric Berberich, Efi Fogel, Dan Halperin, Kurt Mehlhorn, and Ron Wein. Sweeping and Maintaining Two-Dimensional Arrangements on Surfaces: A First Step. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Proceedings of 15th Annual European Symposium on Algorithms (ESA 2007)*, volume 4698 of *LNCS*, pages 645–656. Springer, 2007.

[BFH+09a] Eric Berberich, Efi Fogel, Dan Halperin, Michael Kerber, and Ophir Setter. Arrangements on Parametric Surfaces II: Applications and Concretizations. *Mathematics in Computer Science*, 2009. Submitted.

[BFH+09b] Eric Berberich, Efi Fogel, Dan Halperin, Kurt Mehlhorn, and Ron Wein. Arrangements on Parametric Surfaces I: General Framework and Infrastructure. *Mathematics in Computer Science*, 2009. Submitted.

[BFM+01] Christoph Burnikel, Stefan Funke, Kurt Mehlhorn, Stefan Schirra, and Susanne Schmitt. A Separation Bound for Real Algebraic Expressions. In Friedhelm Meyer auf der Heide, editor, *Proceedings of the 9th Annual European Symposium on Algorithms (ESA 2001)*, volume 2161 of *Lecture Notes in Computer Science*, pages 254–265. Springer, 2001.

[BHK+05] Eric Berberich, Michael Hemmer, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. An Exact, Complete and Efficient Implementation for Computing Planar Maps of Quadric Intersection Curves. In Joseph S. Mitchell and Günter Rote, editors, *Proceedings of the 21st ACM Symposium on Computational Geometry (SoCG 2005)*, pages 99–106. ACM press, 2005.

[BHKT07] Eric Berberich, Michael Hemmer, Menelaos I. Karavelas, and Monique Teillaud. Revision of the Interface Specification of Algebraic Kernel. Technical Report ACS-TR-243301-01, INRIA Sophia-Antipolis, Max-Planck-Institut für Informatik, National University of Athens, 2007.

[BK81] Egbert Brieskorn and Horst Knörrer. *Ebene algebraische Kurven*. Birkhäuser, 1981. In German; English version: Plane Algebraic Curves, Birkhäuser, 1986.

[BK07] Eric Berberich and Lutz Kettner. Linear-time Reordering in a Sweep-line Algorithm for Algebraic Curves Intersecting in a Common Point. Research Report MPI-I-2007-1-001, Max-Planck-Institute for Informatics, Saarbrücken, Germany, 2007.

[BK08] Eric Berberich and Michael Kerber. Exact Arrangements on Tori and Dupin Cyclides. In Eric Haines and Morgan McGuire, editors, *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM 2008)*, pages 59–66, 2008.

[BKS08] Eric Berberich, Michael Kerber, and Michael Sagraloff. Exact Geometric-Topological Analysis of Algebraic Surfaces. In Monique Teillaud, editor, *Proceedings of the 24th ACM Symposium on Computational Geometry (SoCG 2008)*, pages 164–173. ACM press, 2008.

[BKS09] Eric Berberich, Michael Kerber, and Michael Sagraloff. An Efficient Algorithm for the Stratification and Triangulation of Algebraic Surfaces. *Computational Geometry: Theory and Applications*, 2009. Special issue on SoCG 2008. In press.

[BO79] Jon L. Bentley and Thomas Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, 28:643–647, 1979.

[Boe90] Wolfgang Boehm. On Cyclides in Geometric Modeling. *Computer Aided Geometric Design*, 7:243–255, 1990.

[Bos04] Siegfried Bosch. *Algebra*. Springer, 4th edition, 2004.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2nd edition, 2006.

[Bre73] Richard P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973.

[Bre95] Glen E. Bredon. *Topology and Geometry*, volume 139 of *Graduate Texts in Mathematics*. Springer, 1995.

[Bro01] Christopher W. Brown. Improved Projection for Cylindrical Algebraic Decomposition. *Journal of Symbolic Computation*, 32:447–465, 2001.

[Bro02] Christopher W. Brown. Constructing Cylindrical Algebraic Decompositions of the Plane Quickly. Unpublished, 2002.

[BS08] Eric Berberich and Michael Sagraloff. A Generic and Flexible Framework for the Geometrical and Topological Analysis of (Algebraic) Surfaces. In Eric Haines and Morgan McGuire, editors, *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM 2008)*, pages 171–182, 2008.

[BT71] William S. Brown and Joseph F. Traub. On Euclid's Algorithm and the Theory of Subresultants. *Journal of the ACM*, 18:504–514, 1971.

[BT06] Jean-Daniel Boissonnat and Monique Teillaud, editors. *Effective Computational Geometry for Curves and Surfaces*. Springer, 2006.

[Büh95] Katja Bühler. Rationale algebraische Kurven auf Dupinschen Zykliden. Master's thesis, University of Karlsruhe, Germany, 1995. In German.

[CA76] George E. Collins and Alkiviadis G. Akritas. Polynomial Real Root Isolation Using Descartes' Rule of Signs. In Richard D. Jenks, editor, *Proceedings of the 3rd ACM Symposium on Symbolic and Algebraic Computation (SYMSAC 1976)*, pages 272–275. ACM Press, 1976.

[CDH89] Vijaya Chandru, Debasish Dutta, and Christoph M. Hoffmann. On the Geometry of Dupin Cyclides. *The Visual Computer*, 5:277–290, 1989.

[CDR92] John Canny, Bruce Donald, and Eugene K. Ressler. A Rational Rotation Method for Robust Geometric Algorithms. In *Proc. of the 8th Annual Symposium on Computational Geometry (SoCG 1992)*, pages 251–260. ACM press, 1992.

[CGA08] CGAL Editorial Board. *CGAL User and Reference Manual*, 3.4 edition, 2008.

[CGL05] Jin-San Cheng, Xiao-Shan Gao, and Ming Li. Determining the Topology of Real Algebraic Surfaces. In Ralph Martin, Helmut Bez, and Malcolm A. Sabin, editors, *Proceedings of the 11th IMA Conference on the Mathematics of Surfaces*, volume 3604 of *LNCS*, pages 121–146, 2005.

[CGV07] Jorge Caravantes and Laureano González-Vega. Computing the Topology of an Arrangement of Quartics. In Ralph R. Martin, Malcolm A. Sabin, and Joab R. Winkler, editors, *Proceedings of the 12th IMA Conference on the Mathematics of Surfaces*, volume 4647 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2007.

[CJ98] Bob F. Caviness and Jeremy R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Springer, 1998.

[CJK02] George E. Collins, Jeremy R. Johnson, and Werner Krandick. Interval Arithmetic in Cylindrical Algebraic Decomposition. *Journal of Symbolic Computation*, 34:145–157, 2002.

[CLO97] David A. Cox, John B. Little, and Donal O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, 2nd edition, 1997.

[CLO05] David A. Cox, John B. Little, and Donal O'Shea. *Using algebraic geometry*. Undergraduate Texts in Mathematics. Springer, 2nd edition, 2005.

[CLP+09] Jin-San Cheng, Sylvain Lazard, Luis Penaranda, Marc Pouget, Fabrice Rouillier, and Elias Tsigaridas. On the Topology of Planar Algebraic Curves. In John Hershberger and Efi Fogel, editors, *Proceedings of the 25th ACM Symposium on Computational Geometry (SoCG 2009)*, pages 361–370. ACM press, 2009.

[Col67] George E. Collins. Subresultants and Reduced Polynomial Remainder Sequences. *Journal of the ACM*, pages 128–142, 1967.

[Col75] George E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183, 1975. Reprinted in [CJ98], pp. 85–121.

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2nd edition, 2000.

[Dek69] Theodorus J. Dekker. Finding a Zero by Means of Successive Linear Interpolation. In Bruno Dejon and Peter Henrici, editors, *Constructive Aspects of the Fundamental Theorem of Algebra*, pages 37–48. Wiley-Interscience, 1969.

[DET07] Dimitrios Diochnos, Ioannis Emiris, and Elias Tsigaridas. On the Complexity of Real Solving Bivariate Systems. In Christopher W. Brown, editor, *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC 2007)*, pages 127–134. ACM press, 2007.

[DET09] Dimitrios Diochnos, Ioannis Emiris, and Elias Tsigaridas. On the Asymptotic and Practical Complexity of Solving Bivariate Systems over the Reals. *Journal of Symbolic Computation*, 44:818–835, 2009.

[DFMT02] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Algebraic Methods and Arithmetic Filtering for Exact Predicates on Circle Arcs. *Computational Geometry: Theory and Applications*, 22:119–142, 2002.

[DHPS07] Laurent Dupont, Michael Hemmer, Sylvain Petitjean, and Elmar Schömer. Complete, Exact and Efficient Implementation for Computing the Adjacency Graph of an Arrangement of Quadrics. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Proceedings of 15th Annual European Symposium on Algorithms (ESA 2007)*, volume 4698 of *LNCS*, pages 633–644. Springer, 2007.

[DMR08] Daouda Niang Diatta, Bernard Mourrain, and Olivier Ruatta. On the Computation of the Topology of a Non-reduced Implicit Space Curve. In *Proceedings of the 2008 International Symposium on Symbolic and Algebraic Computation (ISSAC 2008)*, pages 47–54. ACM press, 2008.

[Duc00] Lionel Ducos. Optimizations of the Subresultant Algorithm. *Journal of Pure and Applied Algebra*, 145:149–163, 2000.

[Dup22] Charles Dupin. *Applications de Géométrie et de Méchanique*. Bachelier, Paris, 1822.

[EBS09] Pavel Emeliyanenko, Eric Berberich, and Michael Sagraloff. Visualizing Arcs of Implicit Algebraic Curves, Exactly and Fast. In *Proceedings of the 5th International Symposium on Visual Computing (ISVC 2009)*, 2009. accepted.

[EHK+09] Ioannis Emiris, Michael Hemmer, Menelaos Karavelas, Bernard Mourrain, Elias P. Tsigaridas, and Zafeirakis Zafeirakopoulos. Experimental Evaluation and Cross-benchmarking of Univariate Real Solvers. In *Proceedings of the 3rd International Workshop on Symbolic-Numeric Computation (SNC 2009)*, pages 45–54, 2009.

[Eig08] Arno Eigenwillig. *Real Root Isolation for Exact and Approximate Polynomials Using Descartes' Rule of Signs*. PhD thesis, Saarland University, Saarbrücken, Germany, 2008.

[EK08a] Arno Eigenwillig and Michael Kerber. Exact and Efficient 2D-Arrangements of Arbitrary Algebraic Curves. In Shang-Hua Teng, editor, *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 122–131. SIAM, 2008.

[EK08b] Pavel Emeliyanenko and Michael Kerber. An Implementation for the 2D Algebraic Kernel. Technical Report ACS-TR-363602-01, Max-Planck-Institut for Informatics, Saarbrücken, Germany, 2008.

[EK08c] Pavel Emeliyanenko and Michael Kerber. Visualizing and Exploring Planar Algebraic Arrangements – a Web Application. In Monique Teillaud, editor, *Proceedings of the 24th ACM Symposium on Computational Geometry (SoCG 2008)*, pages 224–225. ACM press, 2008.

[EKK+05] Arno Eigenwillig, Lutz Kettner, Werner Krandick, Kurt Mehlhorn, Susanne Schmitt, and Nicola Wolpert. A Descartes Algorithm for Polynomials with Bit-Stream Coefficients. In Victor G. Ganzha, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *8th International Workshop on Computer Algebra in Scientific Computing (CASC 2005)*, volume 3718 of *LNCS*, pages 138–149, 2005.

[EKP+04] Ioannis Z. Emiris, Athanasios Kakargias, Sylvain Pion, Monique Teillaud, and Elias P. Tsigaridas. Towards an Open Curved Kernel. In *Proceedings of the 20th ACM Symposium on Computational Geometry (SoCG 2004)*, pages 438–446. ACM press, 2004.

[EKSW06] Arno Eigenwillig, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. Exact, Efficient and Complete Arrangement Computation for Cubic Curves. *Computational Geometry: Theory and Applications*, 35:36–73, 2006.

[EKW07] Arno Eigenwillig, Michael Kerber, and Nicola Wolpert. Fast and Exact Geometric Analysis of Real Algebraic Plane Curves. In Christopher W. Brown, editor, *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC 2007)*, pages 151–158. ACM press, 2007.

[Eme07] Pavel Emeliyanenko. Visualization of Points and Segments of Real Algebraic Plane Curves. Master's thesis, Saarland University, Saarbrücken, Germany, 2007.

[EST09] Ioannis Emiris, Frank Sottile, and Thorsten Theobald, editors. *Nonlinear Computational Geometry*, volume 151 of *The IMA Volumes in Mathematics and its Applications*. Springer, 2009. to appear.

[ESY08] Arno Eigenwillig, Vikram Sharma, and Chee K. Yap. Almost Tight Recursion Tree Bounds for the Descartes Method. In *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation (ISSAC 2006)*, pages 71–78. ACM press, 2008.

[FGL04] Elisabetta Fortuna, Patrizia M. Gianni, and Domenico Luminati. Algorithmical Determination of the Topology of a Real Algebraic Surface. *Journal of Symbolic Computation*, 38:1551–1567, 2004.

[FGPT03] Elisabetta Fortuna, Patrizia M. Gianni, Paola Parenti, and Carlo Traverso. Algorithms to Compute the Topology of Orientable Real Algebraic Surfaces. *Journal of Symbolic Computation*, 36:343–364, 2003.

[FKMS05] Stefan Funke, Christian Klein, Kurt Mehlhorn, and Susanne Schmitt. Controlled Perturbations for Delaunay Triangulations. In *Proceedings of 16th Annual ACM-SIAM Symposisum on Discrete Algorithms (SODA 2005)*, pages 1047–1056. SIAM, 2005.

[FM02] Stefan Funke and Kurt Mehlhorn. LOOK, a Lazy Object-Oriented Kernel for Geometric Computations. *Computational Geometry: Theory and Applications*, 22:99–118, 2002.

[For12] Andrew R. Forsyth. *Lectures on the Differential Geometry of Curves and Surfaces.* Cambridge University Press, 1912.

[For70] Geoerge E. Forsythe. Pitfalls in Computation, or why a Math Book isn't Enough. *The American Mathematical Monthly*, 77:931–956, 1970.

[For05] Otto Forster. *Analysis 2.* Vieweg, 6th edition, 2005. In German.

[FSH08] Efi Fogel, Ophir Setter, and Dan Halperin. Arrangements of Geodesic Arcs on the Sphere. In Monique Teillaud, editor, *Proceedings of the 24th ACM Symposium on Computational Geometry (SoCG 2008)*, pages 218–219. ACM press, 2008.

[Für07] Martin Fürer. Faster integer multiplication. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 57–66. ACM press, 2007.

[FV96] Steven Fortune and Christopher J. Van Wyk. Static Analysis Yields Efficient Exact Integer Arithmetic for Computational Geometry. *ACM Transactions on Graphics*, 15:223–248, July 1996.

[Gal01] Jean Gallier. Internet Supplement to 'Geometric Methods and Applications for Computer Science and Engineering', Chapter 23: Rational Surfaces. http://www.cis.upenn.edu/~jean/gbooks/geom2.html, 2001.

[GCL92] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1992.

[Gib98] Christopher G. Gibson. *Elementary Geometry of Algebraic Curves.* Cambridge University Press, 1998.

[GLMT05] Gregory Gatellier, Abder Labrouzy, Bernard Mourrain, and Jean-Pierre Técourt. Computing the Topology of 3-dimensional Algebraic Curves. In Tor Dokken and Bert Jüttler, editors, *Computational Methods for Algebraic Spline Surfaces*, pages 27–44. Springer, 2005.

[GO97] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.

[GVEK96] Laureano González-Vega and M'hammed El Kahoui. An Improved Upper Complexity Bound for the Topology Computation of a Real Algebraic Plane Curve. *Journal of Complexity*, 12:527–544, 1996.

[GVN02] Laureano González-Vega and Ioana Necula. Efficient Topology Determination of Implicitly Defined Algebraic Plane Curves. *Computer Aided Geometric Design*, 19:719–743, 2002.

[GVRLR98] Laureano González-Vega, Tomás Recio, Henri Lombardi, and Marie-Françoise Roy. Sturm-Habicht Sequences, Determinants and Real Roots of Univariate Polynomials. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation.* Springer, 1998.

[Hal97] Dan Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press, 1997.

[Hem08] Michael Hemmer. Polynomial. In CGAL Editorial Board, editor, *CGAL User and Reference Manual.* 3.4 edition, 2008.

[HH09] Michael Hemmer and Dominik Hülse. Generic implementation of a modular GCD over Algebraic Extension Fields. In *25th European Workshop on Computational Geometry (EuroCG 2009)*, pages 321–324, 2009.

[HHK+08] Michael Hemmer, Susan Hert, Lutz Kettner, Sylvain Pion, and Stefan Schirra. Number Types. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

[HL03] Dan Halperin and Eran Leiserowitz. Controlled Perturbation for Arrangements of Circles. In *Proceedings of the 19th ACM Symposium on Computational Geometry (SoCG 2003)*, pages 264–273. ACM press, 2003.

[HL07] Michael Hemmer and Sebastian Limbach. Benchmarks on a Generic Univariate Algebraic Kernel. Technical Report ACS-TR-243306-03, Max-Planck-Institute for Informatics, Saarbrücken, Germany, 2007.

[Hof89] Christoph M. Hoffmann. The Problems of Accuracy and Robustness in Geometric Computation. *Computer*, 22:31–40, 1989.

[Hon96] Hoon Hong. An Efficient Method for Analyzing the Topology of Plane Real Algebraic Curves. *Mathematics and Computers in Simulation*, 42:571–582, 1996.

[HS98] Dan Halperin and Christian R. Shelton. A Perturbation Scheme for Spherical Arrangements with Application to Molecular Modeling. *Computational Geometry: Theory and Applications*, 10:273–287, 1998.

[HW07] Iddo Hanniel and Ron Wein. An Exact, Complete and Efficient Computation of Arrangements of Bézier Curves. In Bruno Lévy and Dinesh Manocha, editors, *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling (SPM 2007)*, pages 253–263. ACM press, 2007.

[Joh91] Jeremy R. Johnson. *Algorithms for Polynomial Real Root Isolation*. PhD thesis, Ohio State University, USA, 1991.

[Joh93] John K. Johnstone. A New Intersection Algorithm for Cyclides and Swept Surfaces Using Cycle Decomposition. *Computer Aided Geometric Design*, 10:1–24, 1993.

[Kah08] Mhammed El Kahoui. Topology of Real Algebraic Space Curves. *Journal of Symbolic Computation*, 43:235–258, 2008.

[KC08] Lutz Kettner and Fernando Cacciola. Halfedge Data Structures. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008. Internal Version.

[KCMK00] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. Efficient and Exact Manipulation of Algebraic Points and Curves. *Computer-Aided Design*, 32:649–662, 2000.

[Ker06] Michael Kerber. Analysis of Real Algebraic Plane Curves. Master's thesis, Saarland University, Saarbrücken, Germany, 2006.

[Ker08] Michael Kerber. On Filter Methods in CGAL's 2D Curved Kernel. Technical Report ACS-TR-243404-03, Max-Planck-Institute for Informatics, Saarbrücken, Germany, 2008.

[Ker09a] Michael Kerber. Division-Free Computation of Subresultants Using Bezout Matrices. *International Journal of Computer Mathematics*, 2009. In print.

[Ker09b] Michael Kerber. On the Complexity of Reliable Root Approximation. In Vladimir P. Gerdt, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *11th International Workshop on Computer Algebra in Scientific Computing (CASC 2009)*, volume 5743 of *LNCS*, pages 155–167. Springer, 2009.

[KLPY99] Vijay Karamcheti, Chuanwen S. Li, Igor Pechtchanski, and Chee K. Yap. A Core Library for Robust Numeric and Geometric Computation. In *Proceedings of the 15th Annual ACM Symposium of Computational Geometry (SoCG 1999)*, pages 351–359. ACM press, 1999.

[KM06] Werner Krandick and Kurt Mehlhorn. New bounds for the Descartes method. *Journal of Symbolic Computation*, 41:49–66, 2006.

[KMP+08] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry: Theory and Applications*, 40:61–78, 2008.

[KO63] Anatolii A. Karatsuba and Yu P. Ofman. Multiplication of multidigit numbers on automatata. *Soviet Physics Doklady*, 7:595–596, 1963.

[Kön93] Konrad Königsberger. *Analysis 2*. Springer, 1993. In German.

[KS09] Michael Kerber and Michael Sagraloff. How Complex are Real Algebraic Objects. In *Accepted for the 7th Japan Conference on Computational Geometry and Graphs (JCCGG 2007)*, 2009.

[Lan68] Serge Lang. *Analysis 1*. Addison-Wesley, 1968.

[Lan93] Serge Lang. *Algebra*. Addison-Wesley, 3rd edition, 1993.

[Loo82a] Rüdiger Loos. Computing in Algebraic Extensions. In Bruno Buchberger, George E. Collins, and Rüdiger Loos, editors, *Computer Algebra – Symbolic and Algebraic Computation*, pages 173–188. Springer, 1982.

[Loo82b] Rüdiger Loos. Generalized Polynomial Remainder Sequences. In Bruno Buchberger, George E. Collins, and Rüdiger Loos, editors, *Computer Algebra – Symbolic and Algebraic Computation*, pages 115–138. Springer, 1982.

[LRD00] Henri Lombardi, Marie-Françoise Roy, and Mohad Safey El Din. New Structure Theorem for Subresultants. *Journal of Symbolic Computation*, 29:663–689, 2000.

[LY01] Chen Li and Chee K. Yap. A new Constructive Root Bound for Algebraic Expressions. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA 2001)*, pages 496–505. ACM-SIAM, 2001.

[Mas67] William S. Massey. *Algebraic Topology: An Introduction*. Springer, 1967.

[MC02] Scott McCallum and George E. Collins. Local Box Adjacency Algorithms for Cylindrical Algebraic Decompositions. *Journal of Symbolic Computation*, 33:321–342, 2002.

[McC98] Scott McCallum. An Improved Projection Operation for Cylindrical Algebraic Decomposition. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Springer, 1998.

[McC99] Scott McCallum. Factors of Iterated Resultants and Discriminants. *Journal of Symbolic Computation*, 27:367–385, 1999.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Springer, 1984.

[Mig82] Maurice Mignotte. Identification of Algebraic Numbers. *Journal of Algorithms*, 3:197–204, 1982.

[Mis97] Bhubaneswar Mishra. Computational Real Algebraic Geometry. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 29, pages 537–556. CRC Press, 1997.

[MN00] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 2000.

[Moo79] Ramon E. Moore. *Methods and applications of interval analysis*, volume 2 of *SIAM Studies in Applied Mathematics*. SIAM, 1979.

[MOS06] Kurt Mehlhorn, Ralf Osbild, and Michael Sagraloff. Reliable and Efficient Computational Geometry via Controlled Perturbation. In Michele Bugliesi, Bart Preneel, Vladimir Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, Proceedings Part I (ICALP 2006)*, volume 4051 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2006.

[MS07] Victor Milenkovic and Elisha Sacks. An Approximate Arrangement Algorithm for Semi-algebraic Curves. *International Journal of Computational Geometry and Applications*, 17:175–198, 2007.

[MS09] Kurt Mehlhorn and Michael Sagraloff. Isolating Real Roots of Real Polynomials. In John P. May, editor, *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation (ISSAC 2009)*, pages 247–254. ACM press, 2009.

[Obr03] Nikola Obreshkoff. *Zeros of Polynomials*. Marin Drinov, Sofia, 2003. Translation from the original edition in Bulgarian, 1963.

[Pra90] Michael J. Pratt. Cyclides in Computer Aided Geometric Design. *Computer Aided Geometric Design*, 7:221–242, 1990.

[Pra95] Michael J. Pratt. Cyclides in Computer Aided Geometric Design II. *Computer Aided Geometric Design*, 12:131–152, 1995.

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[PT08] Sylvain Pion and Monique Teillaud. 2D Circular Geometry Kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

[PV07] Simon Plantinga and Gert Vegter. Isotopic Meshing of Implicit Surfaces. *The Visual Computer*, 23:45–58, 2007.

[Raa99] Sigal Raab. Controlled Perturbation for Arrangements of Polyhedral Surfaces with Application to Swept Volumes. In *Proceedings of 15th Annual Symposium on Computational Geometry (SoCG 1999)*, pages 163–172. ACM Press, 1999.

[Rou99] Fabrice Rouillier. Solving Zero-Dimensional Systems Through the Rational Univariate Representation. *Applicable Algebra in Engineering, Communication and Computing*, 9:433–461, 1999.

[Sch34] Issac J. Schoenberg. Zur Abzählung der reellen Wurzeln algebraischer Gleichungen. *Mathematische Zeitschrift*, pages 546–564, 1934. In German.

[Sch85] Arnold Schönhage. Quasi-GCD computations. *Journal of Complexity*, 1:118–137, 1985.

[Sch93] Peter Schorn. An Axiomatic Approach to Robust Geometric Programs . *Journal of Symbolic Computation*, 16:155–165, 1993.

[Sch00] Stefan Schirra. Robustness and precision issues in geometric computation. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of computational geometry*, pages 597–632. North-Holland Publishing Company, 2000.

[SF91] Kevin Suffern and Edward Fackernell. Interval Methods in Computer Graphics. *Computers and Graphics*, 15:331–340, 1991.

[She96] Jonathan R. Shewchuk. Adaptive Precision Floating Point Arithmetic and Fast Robust Geometric Predicates. Technical Report CMU-CS-96-140, Carnegie Mellon University, Pittsburgh, USA, 1996.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971. In German.

[Str06] Adam W. Strzebonski. Cylindrical Algebraic Decomposition using Validated Numerics. *Journal of Symbolic Computation*, 41:1021–1038, 2006.

[SU00] Jörg-Rüdiger Sack and Jorge Urrutia, editors. *Handbook of computational geometry*. North-Holland Publishing Company, 2000.

[Sug99] Kokichi Sugihara. Topology-Oriented Approach to Robust Geometric Computation. In *Algorithms and Computation*, volume 1741 of *LNCS*, pages 357–366. Springer, 1999.

[SW05] Raimund Seidel and Nicola Wolpert. On the Exact Computation of the Topology of Real Algebraic Curves. In Joseph S. Mitchell and Günter Rote, editors, *Proceedings of the 21st ACM Symposium on Computational Geometry (SoCG 2005)*, pages 107–115. ACM press, 2005.

[vdW71] Bartel L. van der Waerden. *Algebra I [früher u.d.T.: Moderne Algebra]*, volume 12 of *Heidelberger Taschenbücher*. Springer, 8th edition, 1971. In German.

[Vin36] A.J. Vincent. Sur la resolution des équations numériques. *Journal de Mathématiques Pures et Appliquées*, 1:341–372, 1836. In French.

[vzGG97] Joachim von zur Gathen and Jürgen Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC 1997)*, pages 40–47. ACM press, 1997.

[vzGG99] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[vzGL03] Joachim von zur Gathen and Thomas Lücking. Subresultants revisited. *Theoretical Computer Science*, 297:199–239, 2003.

[Wal50] Robert J. Walker. *Algebraic Curves*. Princeton University Press, 1950.

[WFZH07] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. Advanced Programming Techniques Applied to CGAL's Arrangement Package. *Computational Geometry: Theory and Applications*, 38:37–63, 2007.

[WFZH08] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. 2D Arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

[Wol] Nicola Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. PhD thesis, Saarland University, Saarbrücken, Germany.

[Wol96] Jürgen Wolfart. *Einführung in die Zahlentheorie und Algebra*. Vieweg, 1996. In German.

[Wol03] Nicola Wolpert. Jacobi Curves: Computing the Exact Topology of Arrangements of Non-Singular Algebraic Curves. In Giuseppe Di Battista and Uri Zwick, editors, *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, volume 2832 of *LNCS*, pages 532–543. Springer, 2003.

[WZ06] Ron Wein and Baruch Zukerman. Exact and Efficient Construction of Planar Arrangements of Circular Arcs and Line Segments with Applications. Technical Report ACS-TR-121200-01, Tel-Aviv University, Tel-Aviv, Israel, 2006.

[Yap97a] Chee K. Yap. Robust Geometric Computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. CRC Press, 1997.

[Yap97b] Chee K. Yap. Towards exact geometric computation. *Computational Geometry: Theory and Applications*, 7:3–23, 1997.

[Yap00] Chee K. Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 2000.

[Yun76] David Y. Yun. On square-free decomposition algorithms. In Richard D. Jenks, editor, *Proceedings of the 3rd ACM Symposium on Symbolic and Algebraic Computation (SYMSAC 1976)*, pages 26–35. ACM Press, 1976.

Index

adaptive precision method, 177, 183
algebraic curve, 20, 80
algebraic number, 19, 55, 100
arc number, 100
arithmetic complexity, 38
arrangement, 93
  overlay, 158, 164, 204, 212
arrangements
  overlay, 196
associates, 18

Bentley-Ottmann sweep-line algorithm, 93, 96–99, 144, 158
Bernstein basis, 51, 75
Bezout's theorem, 24
bit complexity, 38
bitsize
  of a rational, 38
  of a standard interval, 54
  of an integer, 38
bitstream-Descartes method, 74, 116, 148, 153, 200, 206
  Eigenwillig's variant, 74, 148
  m-k-extension, 75, 80–84, 109, 111, 131, 168, 206, 221
  oracle-extension, 84–88, 119, 124, 131, 154
bivariate polynomial, 19
Boolean set operation, 196
bounding box, 99, 200, 217
branch numbers, 102, 109, 113, 163, 208, 224
Brent's method, 59

cluster, 85, 89, 91
cofactor, 207
concept, 157, 189, 196
constructive separation bound, 12, 95, 174, 175
content, 18, 129
coprime, 28, 38, 46, 104, 134, 171
critical x-coordinate, 29, 80, 102, 117
  of a curve pair, 103, 134
critical point, 21, 130
curve analysis, 93, 103, 107–134
curve pair analysis, 93, 104, 134–144
curve-pair-critical, 136
CW-complex, 199
cylindrical algebraic decomposition, 13, 93, 167, 175, 199

Davenport-Mahler bound, 52, 112, 114
dcel, 96, 187, 189, 193
defining polynomial
  of a curve, 20, 94
  of a root, 55, 100, 156
degree
  local, 202
  local gcd, 202
  local real, 202
  of a polynomial, 18
delineability, 22, 25, 100, 202
derivative, 19, 114
Descartes method, 47–54
  complexity analysis, 51–54
  subdivision tree, 49, 51, 79, 81
Descartes' rule of signs, 48, 84
divisor, 18
domain, 18
Ducos' algorithm, 44
Dupin cyclide, 187, 189–192

EGC paradigm, 8, 93, 157, 200
Euclidean division, 39
event point, 118, 133, 224
event queue, 97, 171
event-bounded segments, 121, 124

f-stack, *see* stack, of one curve
factorial domain, 18

fast integer multiplication, 38, 58
Fermat prime, 180
fg-stack, *see* stack, of a curve pair
field of fractions, 19
Fujiwara bound, 48
fully critical, 136
functor, 159

Gauss' lemma, 19, 44
generic position, 95, 107, 133
  of a curve pair, 135, 139
generic programing paradigm, 157, 220
geometric primitives, 8, 97, 105, 144, 157, 158
greatest common divisor, 18
  cofactor, 32
  of polynomials, 30–33, 156

homogenization, 166, 192
Horner scheme, 69

implicit function theorem, 21, 37, 102
inscibed angle theorem, 50, 76
intersection multiplicity, 37, 104, 134, 225
interval
  active, 82, 85
  almost short, 87, 89
  interesting, 85, 89
  isolating, 47, 55, 100
  standard, 54
interval arithmetic, 68–70, 120, 124, 177, 183, 210
irreducible, 18, 21, 55, 200
isocomplex, 17, 93, 118
  stable, 17
isotopic, 16, 93, 199, 213

Jordan curve theorem, 189

Landau's inequality, 41
lazy evaluation, 160, 220

Möbius strip, 187
Möbius transformation, 48
magnitude of polynomial, 39
Mahler measure, 40, 58, 61, 112, 115, 127
minimal polynomial, 55
Minkowski sum, 187
model, 157, 189
modular filter, 156
multiple root, 19
multiplicity, 19, 91
multivariate polynomial, 19

Newton's method, 59
number of sign variations, 34, 76

$\tilde{O}$-notation, 58, 152
Obreshkoff, 50
  area, 50
  disc, 50
  lens, 50, 76
  theorem, 51, 76, 81
one-circle theorem, 50, 77
one-curve-critical, 136, 149
overlay, 164, 218

polynomial evaluation, 41
polynomial remainder sequence (prs), 33
pre-stack, 134
primitive part, 18, 102, 129
primitive polynomial, 18, 202
psc, *see* subresultant, principal coefficient

Rational univariate representation, 167
regular point, 21, 149
resultant, 27, 58, 61, 109, 211
resultant-first strategy, 162, 165
Rolle's theorem, 114
root bound, 22, 48

Schönhage's Theorem, 75
segment of an algebraic curve, 94, 100
segmentation, 24, 93, 102, 199
separation, 52, 66, 78, 142
shear transformation, 17, 36, 108, 117, 135, 140, 152, 154
silhouette, 203
simple root, 19
simplex, 17
simplical complex, 17
singular point, 21, 130
specialization property, 28, 32, 35, 109, 117, 203
speicialization property, 206
square-free, 18, 134, 202
square-free factorization, 46, 137, 168, 171, 180

square-free part, 18, 80, 171, 207
stack
  intermediate, 113–116, 148
  of a *cad*, 214
  of a curve pair, 104, 134, 160, 171
  of one curve, 103, 159, 168, 171
status line, 97, 145
stratification, 199
stratum, 199
strong root isolation, 57–66
strongly critical x-coordinate, 22, 29
structure theorem, 32, 35, 43, 46
Sturm sequence, 34
Sturm's Theorem, 34
Sturm-Habicht sequence, 33, 109, 168, 171, 206
subresultant, 30, 168, 171
  cofactor, 32, 46
  computation, 42–44, 161, 180
  defective, 33
  polynomial, 30
  principal coefficient, 30, 109, 137
  regular, 33
Sylvester (sub)matrix, 27, 30, 153, 226
symbolic computation, 12, 109, 111, 148, 152, 163, 200

Taylor shift, 53, 58, 79, 83
topology computation, 93, 108, 118, 200, 207
torus, 187
total degree, 19
traits class, 157, 189, 196
transit point, 224
trapezoidal map, 226
trivariate polynomial, 19
two-circle theorem, 50, 77

unique factorization domain (UFD), 18
unit, 18
univariate polynomial, 19

vanishing set, 20
vertical component, 21
Voronoi diagram, 187, 197

x-monotone segment, 24, 96, 171

z-fiber, 206

A Realization of Trigonometric Functions

This section describes how approximations of trigonometric functions are computed in Algorithm 5.2. We emphasize that these solutions should not be considered as optimized solutions, but they constitute a certified variant of computing trigonometric functions.

A.1. Approximating π

Given some precision p, our goal is to compute a value pi such that $|\pi - pi| < 2^p$. Despite more modern approaches, we use the rather classical approach of Machin to approximate π. With the formula

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan 1239$$

and the Taylor series for arctan

$$\arctan x = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{2k+1},$$

we can approximate π to any precision, using the expression

$$\pi = \underbrace{\sum_{k=0}^{m-1} \frac{(-1)^k}{2k+1}\left(4\left(\frac{1}{5}\right)^{2k+1} - \left(\frac{1}{239}\right)^{2k+1}\right)}_{=:s_m} + \underbrace{\sum_{k=m}^{\infty} \frac{(-1)^k}{2k+1}\left(4\left(\frac{1}{5}\right)^{2k+1} - \left(\frac{1}{239}\right)^{2k+1}\right)}_{=:r_m}.$$

We proceed in two steps. First, we find $m \in \mathbb{N}$ such that $|r_m| < 2^{p-2}$. Second, we evaluate s_m with interval arithmetic of increasing precision until the width of the resulting interval $I = [c, d]$ is smaller than 2^{p-1}. Then, we return $[c - 2^{p-2}, d + 2^{p-2}]$, which is of a width smaller than 2^p and is guaranteed to contain π.

Finding a suitable m is a technical issue. We have

$$\begin{aligned}
|r_m| &= \left|\sum_{k=m}^{\infty} \frac{(-1)^k}{2k+1}\left(4\left(\frac{1}{5}\right)^{2k+1} - \left(\frac{1}{239}\right)^{2k+1}\right)\right| \\
&\leq \frac{4}{2m+1}\sum_{k=m}^{\infty}\left(\frac{1}{5}\right)^{2k+1} \\
&= \frac{4}{5(2m+1)}\sum_{k=m}^{\infty}\left(\frac{1}{25}\right)^{k} \\
&= \frac{4}{5(2m+1)}\frac{\left(\frac{1}{25}\right)^{m+1}}{\frac{24}{25}} \\
&= \frac{5}{6(2m+1)}\left(\frac{1}{25}\right)^{m+1} \\
&\leq \frac{2^{-4m-4}}{2m+1},
\end{aligned}$$

so if $\frac{2^{-4m-4}}{2m+1} < 2^{p-2}$, or equivalently

$$\log_2 2m+1 > -p-4m-2$$

is satisfied, we have $|r_m| < 2^{p-1}$.

A.2. Approximating sin

We approximate $\sin(\alpha)$ for $\alpha \in [0, \frac{\pi}{4}]$. Note that this range for α suffices for our application. We consider the Taylor series for α. It is well known that

$$\begin{aligned}
\sin(\alpha) &= \sum_{k=0}^{\infty}(-1)^k\frac{x^k}{(2k+1)!} \\
&= \underbrace{\sum_{k=0}^{m-1}(-1)^k\frac{x^k}{(2k+1)!}}_{=:s_m} + \underbrace{\frac{\sin^{(m)}(\xi)}{m!}x^m}_{=:r_m}
\end{aligned}$$

for $\xi \in [0, x]$. As for π, we want to find $m \in \mathbb{N}$ such that $|r_m| < 2^{p-2}$. For simplicity, we derive a global bound that is independent of x, although incorporating x would further improve the bound. Clearly, $|r_m| = \frac{\sin^{(m)}(\xi)}{m!}x^m \leq \frac{1}{m!}$, so if $m! > 2^{-p+2}$, the requirement is fulfilled.

Next, we evaluate s_m using interval arithmetic with increasing precision until the resulting interval is of a width smaller than 2^{p-1}. Then we enlarge the interval on both sides by 2^{p-2} to obtain an interval containing $\sin(\alpha)$.

A.3. Approximating arcsin

The technique to approximate arcsin is analogous to that for sin, but the computation of m is slightly more complicated. Using the Taylor series, we get

$$\begin{aligned}
\arcsin(x) &= \sum_{k=0}^{\infty} \frac{(2k)!}{4^k (k!)^2} \frac{x^{2k+1}}{(2k+1)} \\
&= \underbrace{\sum_{k=0}^{m-1} \frac{(2k)!}{4^k (k!)^2} \frac{x^{2k+1}}{(2k+1)}}_{=:s_m} + \underbrace{\sum_{k=m}^{\infty} \frac{(2k)!}{4^k (k!)^2} \frac{x^{2k+1}}{(2k+1)}}_{=:r_m}
\end{aligned}$$

The bound for m that we derive will depend on x in this case. We have

$$\begin{aligned}
|r_m(x)| &= \sum_{k=m}^{\infty} \underbrace{\frac{(2k)!}{4^k (k!)^2}}_{\leq 1} \frac{x^{2k+1}}{(2m+1)} \\
&\leq \frac{1}{2m+1} \sum_{k=m}^{\infty} x^{2k+1} \\
&= \frac{x}{2m+1} \frac{x^{2m}}{1-x^2}
\end{aligned}$$

So, $|r_m(x)| < 2^{p-2}$ is satisfied if $\frac{x}{2m+1}\frac{x^{2m}}{1-x^2} < 2^{p-2}$, or equivalently

$$\frac{x^{2m+1}}{2m+1} < 2^{p-2}(1-x^2)$$

A global bound can be obtained by exploiting the fact that $0 \leq x \leq \frac{\pi}{4} < \frac{4}{5}$. However, the convergence behavior is much better for x close to zero.

B
Curriculum Vitae

Michael Kerber

Contact Information

Max-Planck-Institut für Informatik
AG1: Algorithms and Complexity
Campus E1.4, Room 327
66123 Saarbrücken, Germany

E-mail: mkerber@mpi-inf.mpg.de
Web: http://www.mpi-inf.mpg.de/∼mkerber

Personal Details

Born February 21, 1981 in Hameln, Germany
German citizenship

Education

10/2006–10/2009	Ph.D. student in Computer Science at the Max-Planck-Institute for Informatics, Saarbrücken, Germany
09/2006	Diplom (Master's degree) in Computer Science at Saarland University, Saarbrücken; Topic: Analysis of Real Algebraic Plane Curves; Advisors: Arno Eigenwillig, Kurt Mehlhorn, Nicola Wolpert
10/2001–09/2006	Undergraduate and master student at Saarland University, Saarbrücken
03/2004	Vordiplom in Mathematics
09/2003	Vordiplom in Computer Science
06/2000	Abitur at Gymnasium am Stadtgarten, Saarlouis

Academic positions

11/2009–	Postdoc at the Institute for Science and Technology Austria (ISTA); Head of group: Herbert Edelsbrunner
03/2005–06/2005	Research Associate at Warwick University, UK; Topic: Fano 3-folds in codimension 4; Hosts: Miles Reid, Gavin Brown
10/2002–03/2009	Teaching Assistant at Saarland University; Beginner and advanced courses in Computer Science and Mathematics; Revision courses for students; Preparation courses for freshmen

Scholarships

11/2006–10/2007	International Max-Planck Research School (IMPRS)

Committees

Member of the Video and Multimedia Presentation Program Committee for the 25th ACM Symposium on Computational Geometry (SoCG 2009)

C List of Publications

Conferences

- Arno Eigenwillig, Michael Kerber, Nicola Wolpert: *Fast and Exact Geometric Analysis of Real Algebraic Plane Curves.* Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC 2007), pp. 151-158.
- Arno Eigenwillig, Michael Kerber: *Exact and Efficient 2D-Arrangements of Arbitrary Algebraic Curves.* Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), pp. 122-131.
- Eric Berberich, Michael Kerber: *Exact arrangements on tori and Dupin cyclides.* Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM 2008), pp. 59-66. An extended abstract was presented as *Arrangements on Surfaces of Genus One: Tori and Dupin Cyclides* at EuroCG 2008.
- Eric Berberich, Michael Kerber, Michael Sagraloff: *Exact Geometric-Topological Analysis of Algebraic Surfaces.* Proceedings of the 24th Annual Symposium on Computational Geometry (SoCG 2008), pp. 164-173. An extended abstract was presented as *Geometric Analysis of Algebraic Surfaces Based on Planar Arrangements* at EuroCG 2008.
- Pavel Emeliyanenko, Michael Kerber: *Visualizing and Exploring Planar Algebraic Arrangements - a Web Application.* Video presented at the 24th Annual Symposium on Computational Geometry (SoCG 2008).
- Michael Kerber: *On the Complexity of Reliable Root Approximation.* 11th International Workshop on Computer Algebra in Scientific Computing (CASC 2009), LNCS 5743, pp. 155-167.
- Michael Kerber, Michael Sagraloff: *How Complex are Real Algebraic Objects?* Accepted for the 7th Japan Conference on Computational Geometry and Graphs (JCCGG 2009).

Journals

- Michael Kerber: *Division-Free Computation of Subresultants Using Bezout Matrices.* International Journal of Computer Mathematics. In print.
- Eric Berberich, Michael Kerber, Michael Sagraloff: *An Efficient Algorithm for the Stratification and Triangulation of an Algebraic Surface.* Computational Geometry: Theory and Applications – SoCG 2008 special issue. In print.

Theses

- Michael Kerber: *Analysis of Real Algebraic Plane Curves.* Diplomarbeit (Master thesis), Saarbrücken, 2006.

Technical Reports

Only those technical reports are listed that have not been published afterwards in a peer-reviewed form.

- Michael Kerber: *On Filter Methods in CGAL's 2D Curved Kernel.* Technical Report ACS-TR-243404-03, Max-Planck-Institute for Informatics, Saatbrücken, 2008.
- Pavel Emeliyanenko, Michael Kerber: *An Implementation for the 2D Algebraic Kernel.* Technical Report ACS-TR-363602-01, Max-Planck-Institute for Informatics, Saatbrücken, 2008.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, 28.09 2009

Michael Kerber