Universität des Saarlandes, FR Informatik
Max-Planck-Institut für Informatik, AG 1

Analysis of Real Algebraic Plane Curves

Diplomarbeit im Fach Informatik
Diploma Thesis in Computer Science

von / by

Michael Kerber

Erstgutachter / first examiner

Prof. Dr. Kurt Mehlhorn

Zweitgutachterin / second examiner

Prof. Dr. Nicola Wolpert

Saarbrücken, 19. September 2006

Hilfsmittelerklärung (Non-plagiarism statement)

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Saarbrücken, 19. September 2006

Acknowledgements

The following persons contributed to the success of this work. I want to thank them all for their support during the last year:

Arno Eigenwillig provided great assistance in all respects, a special thank-you for that. Nicola Wolpert offered me the very interesting topic and was willing to examine this thesis. Kurt Mehlhorn agreed to supervise me. I am especially grateful for the very fruitful discussions with all three named persons.

Eric Berberich and Michael Hemmer additionally assisted me with the software part of this work. I also thank all other EXACUS group members for the excellent code that forms the fundament of my implementation.

Finally, i thank Arno Eigenwillig, Nicola Wolpert and Jens Maue for their useful remarks on earlier drafts of this document.

Abstract

This work describes a new method to compute geometric properties of a real algebraic plane curve of arbitrary degree. These properties contain the topology of the curve as well as the location of singular points and vertical asymptotes. The algorithm is based on the Bitstream Descartes method (Eigenwillig et al.: "A Descartes Algorithm for Polynomials with Bit-Stream Coefficients", LNCS 3718), which computes exact information about the real roots of a polynomial from approximate coefficients. For symbolic calculations with algebraic numbers, especially for counting distinct real roots, it uses Sturm-Habicht sequences (Gonzalez-Vega et al.: "Sturm-Habicht Sequences ...", in: Caviness, Johnson(eds.): *Quantifier Elimination...*, Springer, 1998), which are related to polynomial remainder sequences. Our work explains how to combine these methods to reduce the amount of symbolic calculations without losing exactness.

The geometry of the curve is computed with respect to the predetermined coordinate system. The algorithm changes coordinates in some situations to bring the curve into a generic position, but a new technique transports the computed information back into the original system efficiently. The conditions for a generic position of the curve are less restrictive than in other approaches and can be checked more efficiently during the analysis.

The algorithm has been implemented as part of the software library EXACUS. This work presents comprehensive experimental results. They show that the new approach consistently outperforms the method by Seidel and Wolpert ("On the Exact Computation ...", SCG 2005, 107–115) and the frequently cited algorithm of Gonzalez-Vega and Necula ("Efficient Topology Determination ...", *Comp. Aided Design* **19** (2002) 719–743). We therefore claim that our algorithm reflects the state-of-the-art in the resultant-based analysis of algebraic curves.

Contents

Chapter 1

Introduction

1.1 Problem statement

We present an algorithm for analysing the geometry of an algebraic plane curve. This curve is defined as the vanishing set of $f(x, y)$ where $f \in \mathbb{Z}[x, y]$ is a square free integer polynomial (Figure 1.1.1).

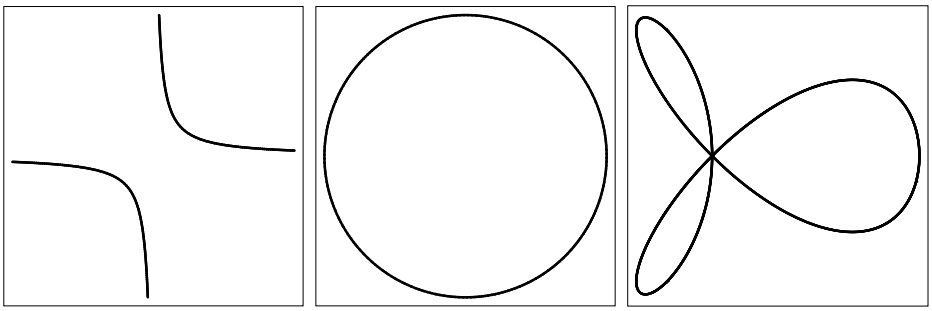

Figure 1.1.1: The curves $xy - 1 = 0$, $x^2 + y^2 - 1 = 0$ and $2x^4 + y^4 - x^3 + xy^2 = 0$.

To understand the outcome of the algorithm, we consider a vertical line, called *sweep line*, that moves from $-\infty$ to $+\infty$ through the plane. In the absence of vertical line components, the sweep line intersects the curve in a finite number of points. We want to know how many intersections take place and where they occur (Figure 1.1.2).

As we can see in Figure 1.1.2, the number of intersections with the sweep line changes at some positions. Our main interest is to detect those points on the curve which are responsible for such a change. We call them *event points* (to be more precise, we also declare singular points of the curve as event points, although some types of singularities do not affect the number of intersections). For the event points, we also count how many arcs of the curve are incident to the point and are entering it from the left side, and how many arcs are entering from the right side

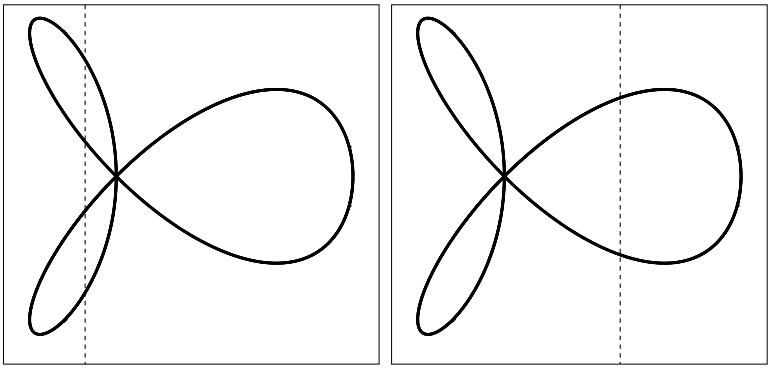

Figure 1.1.2: The sweep line is drawn as dashed line. On the left, we see 4 intersection points, on the right we see 2 intersections.

of the sweep-line. In Figure 1.1.3, we see an example, where the event point in the middle has 4 arcs incident from the left, and 2 arcs incident from the right.

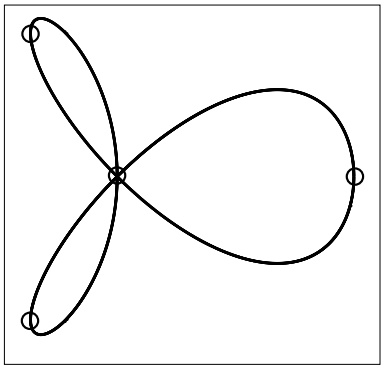

Figure 1.1.3: The event points of an algebraic curve.

There are some special situations which must be detected too: The sweep line reaches a vertical line component of the curve, or in other words, it is completely covered by the curve, and the number of intersections is ∞. We want to know at which positions this happens, and additionally we are interested in the behaviour of the algebraic curve if this vertical line component is removed. Figure 1.1.4 shows an example.

It can also happen that the number of intersections with the sweep line changes, although no event point is responsible for it. This happens in presence of vertical asymptotes of the curve, as depicted in Figure 1.1.5. We are interested in the position of vertical asymptotes and, for each asymptote, how many arcs are converging to it from the left and from the right. Also we want to distinguish whether these arcs go to $+\infty$ or to $-\infty$.

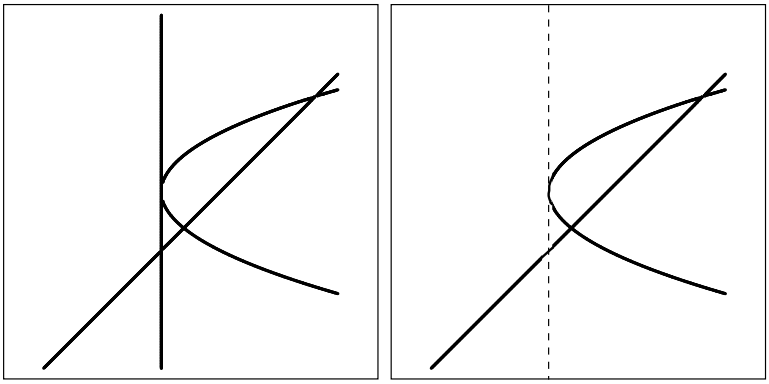

Figure 1.1.4: On the left: A curve with a vertical line component. On the right: If this component is removed, the curve has an event point and a non-event point at this position.

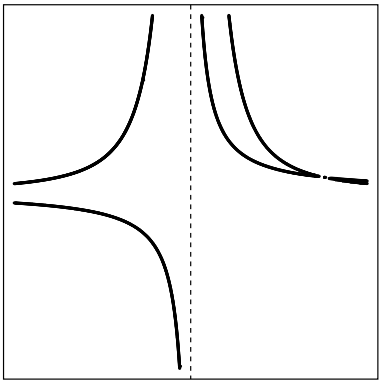

Figure 1.1.5: The dashed line is a vertical asymptote of the curve – there are two asymptotic arcs on the left, and two on the right. On the left, one of these arcs goes to $+\infty$, the other to $-\infty$. On the right, both arc go to $+\infty$.

Let us summarise. For an algebraic curve C, we want to answer the following questions for any x-coordinate $\alpha \in \mathbb{R}$ (α corresponds to the position of the sweep line):

- Does the curve contain the vertical line $x = \alpha$ as a component? If it does, the following questions refer to the curve where the vertical line is removed.
- How many points on the curve C have x-coordinate α? We will call this number n_α.
- How many arcs of the curve converge to the vertical asymptote $x = \alpha$ in direction $+\infty$ and $-\infty$, from the left and from the right?

- For $i \in \{1, \ldots, n_\alpha\}$: Is the point (α, β) an event point, where β is the ith point of C over α (in increasing order)?
- For $i \in \{1, \ldots, n_\alpha\}$: How many arcs are incident to (α, β) from the left, and from the right, where β is defined as above?
- For $i \in \{1, \ldots, n_\alpha\}$ and $\epsilon > 0$: Find an interval containing β of size smaller than ϵ, where β is defined as above.

The problem of answering these questions is interesting in itself because it is the main step in determining the shape of an algebraic plane curve. Furthermore the analysis provides a decomposition of the algebraic curve into x-monotone segments with no singularities in their interiors. This is the first step to compute the arrangements induced by a set of algebraic curves (see references in Section 1.3).

The strategy for the analysis is to decompose the x-axis into cells: *Event x-values* are the x-coordinates of event points, of a vertical line component or of a vertical asymptote – they form singleton cells. The remaining cells are the open intervals between event points. For the event values, a data structure called *vert-line* is created which contains enough information to answer all queries from above concerning that value in constant time, except for the approximation. But the vert-line also contains approximations of the roots which can be refined in an efficient way. Questions concerning non-event numbers can also be answered in constant time (except approximation), once it is known that α is a non-event value.

To get a flavour how complex the geometry of algebraic curves can get, here are some selected examples:

Example. Consider the polynomial

$$f_1 \;=\; (y^4 - 7x^3y^3(x^2 - 3) - 29x^2 - 2)(y^4 + (6x^3 + 2x)y + 7x^2 - 46).$$

For the resulting curve, see the left of Figure 1.1.6. At x-position 0, there are four points on the curve, all non-event points with one incident arc from the left and one incident arc from the right. Up to a precision of $\frac{1}{100}$, the y-coordinates of these points are $-2.60, -1.19, 1.19$ and 2.60.

At x-position $\sqrt{2}$, there are three points on the curve: One point at -19.81 (with precision $\frac{1}{100}$) which is outside the visible region in the graph. A visible point is (near) at -3.11. Both points are non-event points, and with one incident arc from the left and from the right (as any non-event point).

The third point has y-value $\sqrt{2}$. As singularity, this is an event point, and it has two incident arcs from the left and two from the right.

We remark that f_1 has no vertical asymptotes anywhere, although the picture suggest that it does so.

Example. Consider the curve

$$f_2 \;=\; ((x-1)y + 1)(-(x^2 - 1)^2 4y - 3)((4x + 3)^2 + 16y^2 - 1).$$

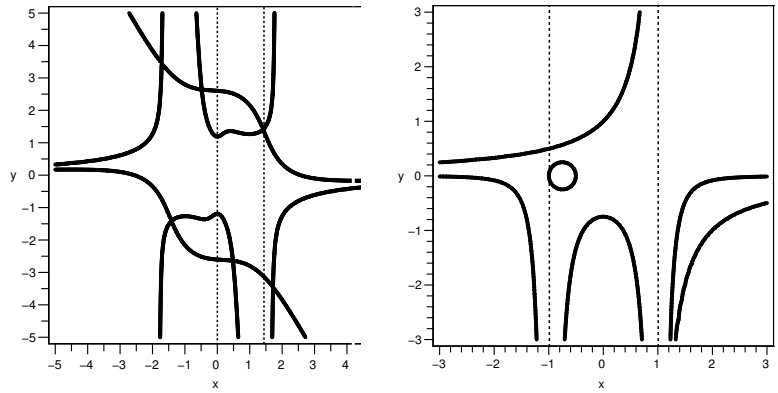

Figure 1.1.6: The graphs of f_1 and f_2.

The graph is drawn at the right in Figure 1.1.6. At x-position -1, the curve has two points. The first one (in increasing order) is an event point with no incident arcs from the left, and two incident arcs from the right. The second one is a non-event point, with one incident arc from the left and from the right. Furthermore, the curve has two arcs converging to the vertical asymptote $x = -1$: one to $-\infty$ from the left, and one to $-\infty$ from the right.

At x-position 1, the curve has no points, and four arcs with vertical asymptote $x = 1$: one to $+\infty$ from the left, one to $-\infty$ from the left and two to $-\infty$ from the right.

Although the two examples above already contain complicated features, they are still "tidy" in some sense, and hence look artificial. In Figure 1.1.7, we show the plots of three curves of total degree 7, 10 and 13. Note that the analysis of these curves is much more difficult (at least for human beings).

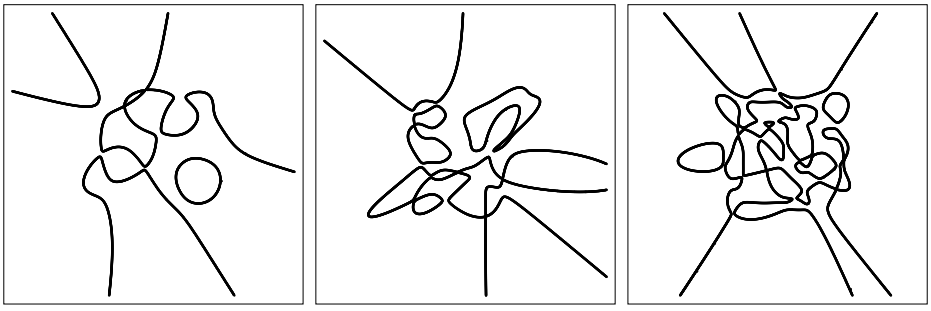

Figure 1.1.7: A gallery of complicated curves.

1.2 Our solution

As already remarked, the goal is to construct for each event value of the curve a vert-line object, containing geometric information about the curve at this x-coordinate. Our algorithm consists of two steps:

- **Projection phase:** Compute a finite set containing all event values of the curve.
- **Extension phase:** For each element α in the computed set, create the vert-line object.

Almost all algorithms for the geometric or topological analysis are subdivided into these two steps. We postpone the discussion of previous work related to our method to Section 1.3.

Event x-values of a curve f are roots of the *resultant* polynomial of f and its derivative with respect to y, $D_y f$. Our projection phase computes this resultant and isolates its real roots using the *Descartes method*. For the extension phase, our algorithm brings in several innovations:

1. Older approaches either involve expensive exact arithmetic over algebraic extensions [ACM84, GK96] or analyse numerically which leads to inexact results in some cases [GN02]. Improvements on exact methods only filter good-natured examples and have to fall back on exact methods in other cases [Ho96, CJK02, Br02]. Our solution manages to apply optimisations for each imaginable input. We combine exact and (controlled) approximate calculations to both achieve exactness and efficiency.

   For exact calculations concerning an event value α (which is irrational in general), we employ *Sturm-Habicht polynomials*, a slight modification of *subresultants*. They deliver the number of points on the curve with x-coordinate α. In most cases, calculating this number brings already enough exact information to complete the extension at α only with approximate calculations. Figure 1.2.1 shows local features of curves that cause additional exact calculations.

   For approximate calculations, we make use of the *Bitstream Descartes method* [EK+05] which isolates the real roots of a square free polynomial as the usual Descartes method, but only by drawing approximations of the coefficients. The remarkable property of this method is that it always computes isolating intervals for the roots of the exact (square free) polynomials without ever touching the exact value of any coefficient. This equips us with an efficient method for root isolation of polynomials over the domain $\mathbb{Z}[\alpha]$. The problem in our work is that most polynomials we explore are not square free. We design and apply several advancements of the (Bitstream) Descartes algorithm that can cope with multiple roots if they are enriched with further information. Mainly, this additive information results from calculations with

the Sturm-Habicht polynomials. So we do not treat these two techniques (Descartes and Sturm-Habicht) as two orthogonal concepts suitable for different substeps of the algorithm, but we interweave them and their interaction improves their performance.

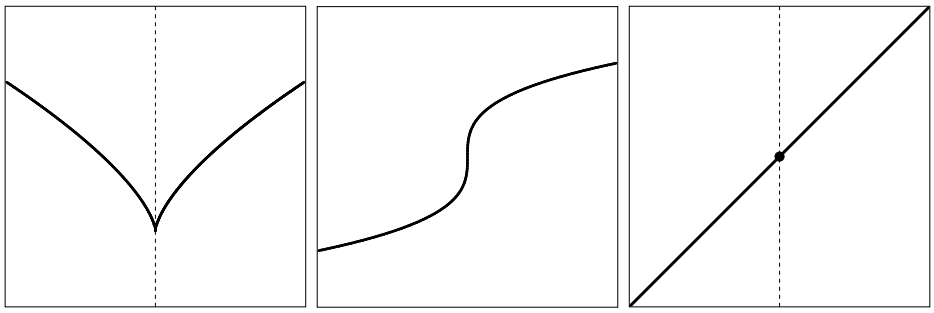

Figure 1.2.1: Three examples of curve features that cause additional symbolic calculations: A vertical cusp, a vertical flex point and an isolated point that lies on a regular point of the curve. The characteristic property is that these points have one arc incident from the left and right, but their partial derivative with respect to y vanishes.

2. Older approaches demand some genericity conditions on the analysed curve. If these conditions are not fulfilled, a different coordinate system is chosen. The geometric computations are then made with respect to this new coordinate system [GK96, GN02, SW05]. We do the same, but we only switch the coordinate system as an intermediate step. A new technique allows to switch back to the original again after the analysis in the transformed system. Consequently, our solution provides an analysis of the input curve in the original coordinate system. That property appears valuable especially in computational geometry because it allows to fix a coordinate system beforehand.

3. We already remarked that we require some genericity condition on the analysed curve and temporarily change coordinates otherwise. Other approaches check whether these conditions are satisfied as a first step in the analysis [GK96, SW05], or the check is distributed over the algorithm [GN02]. Our solution checks genericity as a by-product of the analysis. This means, the algorithm assumes genericity in the first place and detects unfavourable situations where a temporary coordinate change is needed. This is possible because we describe the genericity condition in a "lazy" fashion: On some x-values, we set more genericity conditions than on others.[1] We abandon to give an algebraic description of our term of genericity, but we show that it is bounded by two well-defined genericity assumption on the curve.

[1] In fact, in situations as in Figure 1.2.1, the genericity conditions are most restrictive.

We designed the algorithm with the idea in mind that the vert-line objects shall reflect the true mathematical situation at the corresponding x-coordinates and that all queries shall return the exact result. This matches the *Exact Geometric Computation paradigm* for the building of software libraries: If the algorithm is built upon libraries that commit to this paradigm, then it also commits to the paradigm.

We realised such an implementation as part of the C++ library EXACUS [BE+05, EX], the module is called **AlciX** (**Al**gebraic **c**urves **i**n E**X**ACUS). For basic algorithms which are critical for the running time, such as the approximation of algebraic numbers or the computation of the greatest common divisor, we implemented different variants to find out the optimal performance. With the best configuration, our algorithm clearly outperforms the algorithms `top` from [GN02] and `insulate` from [SW05]. We used `top` with two different initial precisions. Here are the running times of the curves depicted in Figure 1.1.7. Timings are given in seconds:

	AlciX	insulate	top_{60}	top_{500}
degree 7	0.62	7.44	3057	3.60
degree 10	16.5	89.3	>4h	122
degree 13	292	>4h	>4h	2221

We present more running times in Chapter 6.

1.3 Related work

Our problem is a special case of the *cylindrical algebraic decompositions* (cad): In short, for an input set of m polynomials with n indeterminates, a cad is a decomposition of $\mathbb{R}^n$ into cells such that for every cell, the sign of each polynomial is constant. Cads were introduced by Collins [Co75] for quantifier elimination, a general algorithm for the cad computation can be found in [ACM84]. This algorithm also consists of a projection and an extension phase. More precisely, their approach also includes a base phase because the projection must be performed several times in higher dimension. For the plane, the base phase can be integrated into the projection phase for a simpler description. The extension phase is realised with expensive computations over algebraic extensions and modern approaches try to reduce the costs of the extension phase. Our algorithm constitutes an improvement for $m = 1$ and $n = 2$. Collins et al. [CJK02] described how interval arithmetic can speed up the extension phase, at least in favourable situations, by using interval arithmetic for the Descartes method. We also mention an unpublished manuscript from Brown [Br02]: Even though the Descartes algorithm with interval arithmetic fails for non-square free polynomials, he can still sometimes benefit from its output, thanks to additional knowledge of the geometric situation. However, this improvements only apply to easy cases. Moreover, both methods ([CJK02, Br02]) can fail in case that the coefficients are not approximated precisely enough, so their

improvements include no success guarantee. In case of a failure, one must fall back to a (slow) exact method. The improvements of our solution apply for each polynomial no matter of the level of degeneracy.

A closely related problem is *topology computation*: For a given algebraic curve C, a graph homeomorphic to C is computed. Almost all algorithm for topology computation also divide into projection and extension phases.[2] We will discuss recent results in this area and focus on the extension step: The algorithm from Gonzalez-Vega and Necula [GN02] uses Sturm-Habicht sequences to ensure that the x-values of event points are pairwise disjoint, otherwise it changes the coordinate system. The y-value of the event point can be written as a rational expression in terms of α_i. This allows to explicitly divide out (numerically) the non-square-free part of $f(\alpha_i, y)$ and to isolate its real roots.

The authors describe their algorithm as "seminumerical", i.e. several computation steps are performed with floating point numbers. Therefore, the algorithm can return wrong results for ill-natured inputs. However, we adopt some methods from this work, such as the Sturm-Habicht sequences and the rational expression for the y-value for our algorithm. The main difference to our approach lies in the replacement of purely numerical methods by the Bitstream Descartes algorithm to maintain exactness. Also, we save costs of intermediate calculations by introducing new variants of the Descartes method and by avoiding genericity checks. An additional feature of our work is that we transport the result back into the original coordinate system.

We also refer to the work of Gonzalez-Vega and El Kahoui [GK96]. Their algorithm contains the same techniques as [GN02] but with two main differences. For the representation of an algebraic number α, it uses *Thom's codes*, the sign sequence of the derivatives of some polynomial for which α vanishes. Therefore, it does not approximate to floating point numbers and produces the exact result. Also, it checks for generic position beforehand and applies a deterministic method to find a generic direction. The authors present the running time of $O(n^{16}(\log n)^5)$, where n is the maximum of the degree and the maximal bit length of the coefficients of the polynomial. This running time seems to be the best known complexity bound for the problem.

Hong [Ho96] proposes a solution using Sturm sequences: He derives a Sturm sequence for $f(\alpha_i, y)$ from subresultant remainder sequences and uses the Sturm method to isolate the roots of the polynomial. All roots over α_i are then encapsulated in boxes and the number of arcs running into the points is computed by intersecting the curve with the boundaries of the boxes. The solution can cope with several event points over α_i, but it assumes y-regularity of the curve. In other words, it cannot handle vertical asymptotes. As it uses Sturm sequences, the polynomials need not to be made square free before isolation. However, for the sign variation count during the Sturm isolation, one needs to evaluate repeatedly the

[2] A remarkable exception is [CF+05], where Gröbner basis methods are used

sign of numbers in $\mathbb{R}[\alpha_i]$. This step also involves expensive symbolic calculations in the worst case. Hong uses floating point approximation and interval arithmetic to filter easy cases and reports a good practical behaviour.

Most recently Seidel and Wolpert [SW05] came up with a solution, that, as they say, "exploits a little more geometry and a little less algebra". Their idea is very intuitive: By computing the discriminant also with respect to the x- and y-variable and isolating their roots, they end up with a quadratic number of boxes that contain all event points. Now, a third direction is chosen and the discriminant is computed with respect to this direction. More precisely, the coordinate system is transformed such that the chosen direction becomes the y-axis, and the discriminant with respect to y is evaluated. This third discriminant eliminates the boxes that do not contain event points if the third direction is chosen luckily (which happens with high probability). Once the correct boxes are found, they are shrunk further until they contain only one root of $f(\alpha_i, y)$, and the number of incident arcs can be obtained similar as in Hong's approach.

This method also impose genericity conditions on the input curve that must be checked with algebraic computations in an initial step. The shrinkage of the boxes gives rise to further expensive calculations. On the other hand, this article indicates how to extend its idea to the case of several curves.

For older approaches for topology computation, see the references of [GN02, GK96, Ho96, SW05].

The resultant of two polynomials is a standard object in elimination theory [CLO92] from the nineteenth century. It is defined as the determinant of the Sylvester matrix. The subresultants are defined as polynomials whose coefficients are minors of the Sylvester matrix. Already Sylvester was aware of this concept [Sy40], and Habicht [Ha48] used them for creating Sturm sequences. Collins [Co67] and Brown and Traub [BT71] point out their relationship with the polynomials from the Euclidean algorithm. For a historical survey, we recommend [GL03]. Subresultants have also become textbook material in the last years [GCL92, Yap00, BPR03]. Ducos [Du00] and Lombardi et al. [LRS00] presented the best known algorithms for computing the subresultants.

Sturm-Habicht polynomials are introduced in [GL+98]. Generally, they are defined for two polynomials f and g to produce a "Sturm-like" sequence for f and $f'g$. However, we will not make use of them in full generality, but we will restrict to the case $g = 1$ (see also [GN02]). Then, the Sturm-Habicht polynomials correspond to the subresultants of f with its derivative, but with some sign changes. These negations bring the sequence of Sturm-Habicht polynomials nearer to a Sturm sequence, but it is not a Sturm sequence in general. However, the sequence can be used to compute the number of real roots of f and has good specialisation properties.

The Descartes method to isolate real roots is described by Collins and Akritas [CA76]. The idea is to get an upper bound for the number of roots in some inter-

val, using Descartes' rule of sign. If this bound is greater than one, the interval is subdivided. The running time of the algorithm depends on the recursion depth, Eigenwillig et al. [ESY06] have recently presented the best known bound. Johnson and Krandick [JK97] demonstrate that, in favourable situations, double precision arithmetic for the coefficients suffice to isolate real roots, [CJK02] extends the idea by allowing arbitrarily high precisions, but only up to an a priori specified bound. This enlarges the class of favourable inputs but still leaves unlucky ones where the algorithm fails. Eigenwillig et al. [EK+05] improved on this result with the Bitstream Descartes method which is able to handle all sorts of input polynomials only by approximations. This algorithm illustrates the idea of calculating approximately with guarantees very nicely: Outwards, it always delivers valid isolating intervals for the real roots of the (exact) input polynomial. But inside, the algorithm never looks at the exact value of the coefficients. Doubtful decisions during the algorithm are prevented through a randomised choice of split points in the subdivision step, and by increasing the approximation when necessary.

The Exact Geometric Computation paradigm is described in [YD95]. The idea behind is that the interface of the library always returns the exactly right result. This means that, for instance, rounding errors are not allowed. However, it is not specified how the internals of the library should look like: If it is guaranteed that the right decision can be taken by approximate (and inexact) calculation, this can improve the performance. The challenge is to reduce the cost of computing the correct result as much as possible. The paradigm has been used successfully in the libraries LEDA [MN00, LE] and CGAL [FG+98, CG], mainly for linear objects. The latter also contains algorithms to compute arrangements of line segments, polylines, conic arcs and arcs of rational functions [CG]. An overview about applications of arrangements can be found in [AS00] and [Ha04].

EXACUS [BE+05, EX] is another library for geometric problems dealing with higher degree objects. Arrangements computation of conics [BE+02] is available, methods for arrangements of cubic plane curves [EK+06] and for arrangements of quadrics in space [BH+05] were also implemented. All these methods use a generic sweep line algorithm that requires a collection of predicates for a single curve and a pair of curves (see [BE+05]). For the predicates only concerning one curve, an analysis of algebraic curves is performed, with a similar outcome as our algorithm. These algorithms, however, rely on special properties of the considered curves of bounded degree. For instance, a singular points of a conic is always an intersection of two line components. The algorithms are therefore not directly transferable to arbitrary curves. Though we point out that our viewpoint of the problem for analysing curves is inspired by previous work in EXACUS and we tried to snap in our terminology with respect to these works.

1.4 Overview

We start in Chapter 2 by establishing the mathematical tools needed for the algorithm. Chapter 3 describes the algorithm in detail except the extension phase (i.e. the analysis of the curve at certain x-values), which is explained in the two subsequent chapters. For curves satisfying certain genericity assumptions, the extension phase is treated in Chapter 4, whereas Chapter 5 handles arbitrary plane curves. In Chapter 6 we discuss implementation details and present experimental results.

There are two reason why we spend two chapters on the extension phase: First of all, the method for generic curve is more efficient (if it works) because it reduces the number of symbolic computations. Furthermore, the division improves the presentation: Discussing generic curves first allows to encapsulate techniques in a simpler setting leading to a yet incomplete method. Combining these techniques with additional ideas for arbitrary curves finally leads to the complete solution.

Chapter 2

Mathematical Foundations

We start with a discussion of fundamental algebraic definitions and results in Section 2.1. Most of the presented material should be known and is included to put the remaining sections on a stable basis. In Section 2.2, we define the geometric objects which are in the focus of this work and we show how geometric properties of algebraic plane curves are related to the algebra of bivariate polynomials. In Section 2.3, we turn to univariate polynomials, especially how to find their real roots in an efficient way, using the so called *Descartes method*. How these real roots can be represented is the subject of Section 2.4. We also discuss which basic operations can be implemented efficiently in this representation. Subresultants, the subject of Section 2.5 are closely related to the intermediate results in the Euclidean algorithm and provide rich information about a pair of polynomials. The Sturm-Habicht polynomials are certain specialisations of subresultants and allow to count the number of real roots of polynomials, as we explain in Section 2.6.

2.1 Algebra

This section summarises the definitions and results from basic algebra we need in this work. We tried to restrict to material which can be easily found in introducing textbooks in algebra (as [La93, Bo01, Wo96, Wa71]). So, the experienced reader should be able to go quickly through this section. We assume that the reader is familiar with very basic concepts from algebra like polynomial rings and fields. For each ring, we demand commutativity and the existence of a unity element.

2.1.1 Domains

We start with some basic definitions. A domain D is a ring with 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 $a = ub$ with $u \in D$ a unit. Being associate is an equivalence relation; we define $\mathrm{Rep}(D)$ to be a representative system for the equivalence classes of D. We assume that 1 is always chosen as representative for the units of D. For some familiar domains, there are canonical choices for all representatives:

Example.

- For $D = \mathbb{Z}$, the units are 1 and -1. The canonical choice for $\mathrm{Rep}(\mathbb{Z})$ are the natural numbers.
- For a field K, set $D = K[t]$. The units are precisely the non-zero constants. The canonical choice for $\mathrm{Rep}(D)$ are the monic polynomials, i.e. polynomials with leading coefficient 1, plus the zero element.

The next definition should be familiar to everybody, at least for integers:

Definition 2.1.1. Let $a, b \in D$. A *greatest common divisor* of a and b is an element $g \in D$ such that

- g divides a and g divides b and
- any other element dividing both a and b is divisor of g.

In case of existence, all greatest common divisors of a and b are associates. Therefore, there is a unique greatest common divisor in $\mathrm{Rep}(D)$ and we call it *the greatest common divisor* of a and b, in signs $\gcd(a, b)$.

Note that $\gcd(a, b)$ need not exist for arbitrary domains. We need a bit more structure to ensure the existence of greatest common divisors, as well as for other properties. We call a non-unit $a \in D$ *irreducible*, if for each decomposition $a = bc$ either b or c is a unit. With that definition, it is clear that each element can be decomposed into irreducible elements. But this decomposition is not necessarily unique – this motivates the next definition.

Definition 2.1.2. A domain D is *factorial*, if each element $r \neq 0$ of D can be uniquely (up to order) decomposed into

$$r = u \cdot p_1 \cdots p_r$$

with u a unit and each $p_1, \ldots, p_r \in \mathrm{Rep}(D)$ irreducible.

A factorial domain is also called *unique factorisation domain* or just *UFD*.

Which domains are factorial? First of all, a field is trivially factorial. Also, $\mathbb{Z}$ is factorial because each integer can be decomposed uniquely into prime numbers (and they are irreducible by definition). The following theorem is usually called "Gauss Theorem", see [La93, §II.3] or [Bo01, §2.7] for proofs.

Theorem 2.1.3 (Gauss). If D is factorial, then $D[t]$ is factorial.

It follows inductively that each polynomial ring over a field or over $\mathbb{Z}$ is factorial.

A characteristic property of factorial rings is that irreducible elements are *prime*, which is stated in the next lemma

Lemma 2.1.4. Let D be factorial. If the irreducible $a \in D$ divides the product bc, it divides b or it divides c.

Proof. If a divides bc, we have that $ad = bc$ for some $d \in D$. Now, we can decompose both sides into irreducible elements, since D is factorial:

$$u_1 a d_1 \cdot \ldots \cdot d_l = u_2 b_1 \cdot \ldots \cdot b_m c_1 \cdot \ldots \cdot c_n$$

and from the uniqueness of the decompositions, we know that a is associate to some b_i or some c_j and the result follows. □

The next lemma shows that the gcd is well-defined over factorial rings.

Lemma 2.1.5. For D factorial and $a, b \in D$, $\gcd(a, b)$ exists.

Proof. For $c \in D$ and $p \in D$ irreducible, let $\mu_p(c) := \max\{k \mid p^k \text{ divides } c\}$ be the *multiplicity* of the factor p in c. Let $a_1, \ldots, a_r$ be the distinct irreducible factors of a. Set $e_i := \min\{\mu_{a_i}(a), \mu_{a_i}(b)\}$ and define

$$g := \prod_{i=1}^{r} a_i^{e_i}$$

Clearly, g is a common divisor of a and b. Now, any $h \in D$ that does not divide g has either some irreducible factor which is not equal to one a_i, or it contains some a_i more often than g. In both cases, it cannot be a divisor of a and b. □

We can naturally extend the definition of the gcd to any finite set of elements:

Definition 2.1.6. For $a_1, \ldots, a_r \in D$ with $r \geq 3$, we define recursively

$$\gcd(a_1, \ldots, a_r) = \gcd(a_1, \gcd(a_2, \ldots, a_r))$$

For a factorial domain D, this value is well-defined.

2.1.2 Polynomial rings

We assume from now that D is factorial and consider the polynomial ring $D[t]$ which is also factorial by Theorem 2.1.3. As we have defined greatest common divisors for finite sets of elements, we can define:

Definition 2.1.7. For $f = \sum_{i=0}^{n} a_i t^i \in D[t]$, we define the *content* of f to be the gcd of the coefficients, $\operatorname{cont}(f) = \gcd(a_0, \ldots, a_n)$. A polynomial is called *primitive*, if $\operatorname{cont}(f) = 1$. The polynomial

$$\operatorname{pp}(f) := \frac{f}{\operatorname{cont}(f)}$$

is called the *primitive part* of f.

As we will see later, the content has a simple geometric meaning for bivariate polynomials.

Our next goal is to define what it means for a polynomial in $D[t]$ to be *square free*. However, we make a small detour for this: We first define square free polynomials for $K[t]$ with K a field. First of all, we rephrase the fact that $K[t]$ is factorial:

Proposition 2.1.8. Each $f \in K[t]$ can be decomposed uniquely as

$$f = u \cdot f_1 \cdot \ldots \cdot f_r$$

where $u \in K$ is the leading coefficient of f and $f_1, \ldots, f_r$ are monic irreducible polynomials in $K[t]$ of positive degree.

Now, we define square freeness in $K[t]$:

Definition 2.1.9. A polynomial $f \in K[t]$ is *square free*, if all f_i's from Proposition 2.1.8 are distinct.

To extend this definition to the polynomial ring $D[t]$, we introduce the concept of a *quotient field*:

Definition 2.1.10. For a domain D, we define

$$Q(D) := \{(a, b) \mid a \in D, b \in D - \{0\}\} / \sim$$

with $(a, b) \sim (c, d)$ if and only if $ad = bc$. $Q(D)$ is called *the quotient field of* D. Indeed, with the addition and multiplication defined as

$$\begin{aligned} (a, b) + (c, d) &= (ad + bc, bd) \\ (a, b) \cdot (c, d) &= (ac, bd) \end{aligned}$$

$Q(D)$ forms a field, the smallest field containing the domain D. The elements of $Q(D)$ are written as fractions $\frac{a}{b}$, where a is called the *numerator* and b the *denominator*.

As an example, $Q(\mathbb{Z}) = \mathbb{Q}$. In fact, this is the only example we need in this work.

From now on, we write $Q := Q(D)$. We can naturally embed elements of $D[t]$ into $Q[t]$ by embedding the coefficients. We will say that a polynomial in $D[x]$ is square free if it is square free over $Q[x]$.

We can give a number of equivalent descriptions for square freeness. We remark that polynomials are formally differentiable with the familiar rules from calculus and we write f' for the derivative of f.

Theorem 2.1.11. Let $f \in D[t]$. The following conditions are equivalent:

1. f is square free.

2. For each non-constant $g \in Q[t]$, g^2 does not divide f. (In other words, f contains no squares.)

3. f and f' have no common divisor except constants, i.e. $\gcd(f, f')$ is a constant.

Proof. We use contraposition in each step:
$1 \Rightarrow 2$: we assume that there is some g such that g^2 divides f, so $hg^2 = f$ for some h. Decomposing each side (considered as polynomial over Q) into irreducible components, it follows that each irreducible factor of g appears twice in f. So f is not square free.
$2 \Rightarrow 3$: Assume that f and f' have a common factor h. W.l.o.g. we can assume that $h \in \mathrm{Rep}(D[t])$ and irreducible. Decompose

$$f = uf_1 \cdot \ldots \cdot f_r$$

into monic irreducible factors over Q. W.l.o.g. we can assume that $h = f_1$. By the rules of the derivative, we have that

$$f' = u(h(f_2 \cdot \ldots \cdot f_r)' + h'(f_2 \cdot \ldots \cdot f_r))$$

and since h divides this expression, it follows that h divides the product $h'(f_2 \cdot \ldots \cdot f_r)$. From Lemma 2.1.4, we know that h must divide h' or one of the f_i's. The former is impossible because h is not constant. Thus w.l.o.g. h divides f_2, and since both are irreducible and in $\mathrm{Rep}(D)$, it already holds that $h = f_2$. Therefore, h^2 divides f.
$3 \Rightarrow 1$: If f is not square free, it immediately follows that $f = g^2 \cdot h$ for some irreducible factor g. The derivative of f is

$$f' = g^2h' + 2gg'h$$

and we see that g divides f'. Therefore, g is a common divisor of f and f'. □

We remark the obvious corollary that irreducible polynomials are square free.

2.1.3 Roots of polynomials

A *root* r of a polynomial f is an element for which $f(r) = 0$. If $f \in D[t]$, then it does not necessarily have a root in D (for instance, $D = \mathbb{Z}$ and $f = 2x - 1$). But also if we pass to $Q = Q(D)$, there is not always a root ($f = x^2 - 2$ has no rational root). We need another concept to ensure that there is always a root of f:

Definition 2.1.12. A field K is called *algebraically closed* if each non-constant polynomial in $K[t]$ has a root in K. In particular, each non-constant polynomial factorises into linear factors.

The most famous example of an algebraically closed field are the complex numbers.

Definition 2.1.13. Let $D \subset L$ with D a domain and Ł a field, and $\alpha \in L$. α is called *algebraic* over D, if there exists a polynomial $f \in D[t]$ such that $f(\alpha) = 0$.

If a number is algebraic over Q, it is also algebraic over D since the polynomial over Q can be multiplied by a constant such that all denominators vanish, and the result is an element of $D[x]$ with the same roots. Passing to the field Q does not enlarge the set of algebraic numbers, consequently.

Definition 2.1.14. Let $K \subset L$ be fields. If each element in L is algebraic over K, we call Ł an *algebraic extension* of K.

Remark. For instance, the extension $\mathbb{Q} \subset \mathbb{R}$ is not algebraic since there exists real numbers which are not algebraic (they are also called *transcendental*). This is easy to see, since the algebraic numbers over $\mathbb{Q}$ are countable, but $\mathbb{R}$ is not countable.

The next theorem guarantees that we can always find roots of polynomials if we embed the domain into a suitable field:

Theorem 2.1.15. For a field K, there exists an algebraically closed field $\overline{K}$ containing K such that $K \subset \overline{K}$ is an algebraic extension. $\overline{K}$ is called an *algebraic closure* of K.

See [La93, §V.2],[Bo01, §3.4] for proofs, or [Wo96, 6.3.3] for a simplified and weaker version. The algebraic closure is only unique up to an isomorphism (which restricts to the identity on K) and there is no canonical choice.

From now, we set $C := \overline{Q} = \overline{Q(D)}$, where we choose one algebraic closure arbitrarily. An equivalent description for algebraically closed fields is that the irreducible elements of $C[t]$ are precisely the polynomials of degree 1. The following proposition is therefore clear:

Corollary 2.1.16. Each polynomial $f \in Q[t]$ can be written uniquely as a product of linear factors:

$$f = u \prod_{i=1}^{n} (t - \alpha_i)$$

with u the leading coefficient of f and $\alpha_i \in C$.

A polynomial of degree n has n roots over C, but some of them can occur several times.

Definition 2.1.17. Let $f \in Q[t]$ with root $\alpha \in C$. The *multiplicity* m of α is the number of linear factors $(t - \alpha)$ of f. A root is called *simple* if its multiplicity is one, and *multiple* otherwise. A root with multiplicity k is also called a *k-fold root*.

For example, 0 is a root for $x^3 - 2x^2$ of multiplicity 2.

Let K be a field. It is well-known that the greatest common divisor of two polynomials f and g over $K[t]$ is obtained with the *Euclidean Algorithm*:

Algorithm 2.1.18 (Euclid).
Input: $f, g \in K[t]$
Output: $\gcd(f, g)$

1. If $g = 0$, return f.
2. Write $f = qg + r$ with $\deg r < \deg g$.
3. Compute $\gcd(g, r)$.

The intermediate polynomials in the algorithm never leave the field K, so it follows at once:

Proposition 2.1.19. Let $f, g \in D[t]$ be polynomials. Let h be the greatest common divisor of f and g, considered as polynomials in $Q[t]$, and h^* their greatest common divisor over $C[t]$. Then $h = h^*$.

This allows the proof of another characterisation of square freeness:

Theorem 2.1.20. A polynomial $f \in Q[t]$ is square free if and only if all roots of f over C are simple.

Proof. Consider a multiple root α of f. It follows that $(x - \alpha)^2$ divides f. Consequently, $(x - \alpha)$ divides f'. So, f and f' have a non-constant common divisor in $C[t]$ and it follows they also have a non-constant common divisor in $Q[t]$.

On the other hand, if f is not square free, a component appears twice, and each root of this component is multiple. □

2.2 Geometry

We introduce the main geometric objects for our approach, like event points, arcs of a curve, vertical asymptotes etc. For all those, we derive a description in terms of algebra and we explore the properties that are relevant for our purposes. Some of the results are also valid in a more general setting, but the theory is only derived as much as necessary.

Our viewpoint of algebraic curves is inspired by the previous works in the EXACUS project, for instance [EK+06], [BH+05].

2.2.1 Algebraic, square-free and primitive curves

Throughout this section, we consider bivariate integer polynomials $f \in \mathbb{Z}[x, y]$. For a monomial $x^i y^j$, we define its *degree* to be $i + j$. For a general polynomial f, the *total degree* $\deg f$ is the maximal degree of all its monomial terms. The *y-degree* of f, $\deg_y f$, is the degree of f considered as a polynomial in y.

We start by defining the central object of our interest, the algebraic curve:

Definition 2.2.1. Let $\mathbb{K}$ be the field $\mathbb{R}$ or $\mathbb{C}$, and $f \in \mathbb{Z}[x, y]$. We define the *algebraic curve* induced by f to be the point set

$$C_{\mathbb{K}}(f) := \{(x, y) \in \mathbb{K}^2 \mid f(x, y) = 0\}$$

If $\mathbb{K} = \mathbb{R}$, we call $C_{\mathbb{R}}(f)$ a real algebraic curve, if $\mathbb{K} = \mathbb{C}$, we call $C_{\mathbb{C}}(f)$ a complex algebraic curve.

Let $f = f_1 \cdot f_2$ be a decomposition. The curves induced by f_1 and f_2 are called *components* of the curve induced by f. Indeed, it is not hard to verify that

$$C_{\mathbb{K}}(f) = C_{\mathbb{K}}(f_1) \cup C_{\mathbb{K}}(f_2)$$

and this also implies that $C_{\mathbb{K}}(f) = C_{\mathbb{K}}(f_1^k f_2)$. Hence different polynomials can induce the same algebraic curve.

From now, for the sake of brevity, we will say *the curve* f instead of the real algebraic curve induced by the polynomial f. In the same way, we write a *point on the curve* for a point in the real plane that solves the defining equation $f = 0$, and we explicitly emphasise when we talk about complex curves and points.

We interpret $f \in \mathbb{Z}[x, y]$ as polynomial in y, with coefficients in the domain $\mathbb{Z}[x]$ for the moment. This allows us to talk about the content and the primitive part of f, as defined in Definition 2.1.7. In other words, we can decompose $f = \text{cont}(f)\text{pp}(f)$ with $\text{cont}(f) \in \mathbb{Z}[x]$, $\text{pp}(f) \in (\mathbb{Z}[x])[y]$. We now define square free polynomials over bivariate integer polynomials. Recall the definition for univariate polynomials and its equivalent formulations from Section 2.1.2.

Definition 2.2.2. $f \in \mathbb{Z}[x, y]$ is called *square free* if $\text{cont}(f)$ is square free and $\text{pp}(f)$ considered as polynomial in y with coefficients in $\mathbb{Z}[x]$ is square free.

We show that this definition is indeed equivalent to the property that f does not contain a non-constant square.

Lemma 2.2.3. f is square free if and only if for each non-constant polynomial $g \in \mathbb{Z}[x, y]$, g^2 does not divide f.

Proof. W.l.o.g. we can restrict to irreducible polynomials g. If f is not square free, then either $\text{cont}(f)$ or $\text{pp}(f)$ has a non-constant square as divisor. For the other direction, assume that such a polynomial g exists with g^2 divisor of f. There are two cases: If g does not contain the variable y, then neither does g^2. So g^2 must be divisor of the content of f. If g does contain y, then it cannot divide the content, and as irreducible element, it must divide the primitive part. So, also g^2 must divide $\text{pp}(f)$. □

The curve f consists of the irreducible components of $\text{cont}(f)$ together with the irreducible components of $\text{pp}(f)$. Containing a component twice does not change the curve by definition. Therefore, we can throw away multiple components without loosing the ability to represent each algebraic curve:

Theorem 2.2.4. For each algebraic curve C, there is a square free polynomial f with $C = C_{\mathbb{K}}(f)$.

Proof. Let C be induced by some polynomial f. If f is square free, we are done. Otherwise, there exists a non constant g such that g^2 divides f. We define $\tilde{f} := \frac{f}{g}$. This decreasess the total degree of f by at least one, and $\tilde{f}$ and f define the same algebraic curve. We can repeat in this manner until we obtain a square free polynomial. This happens after finitely many steps as the degree can only decrease finitely many times. □

The content has a very direct meaning in terms of geometry. We define formally what it means for the curve f to have a vertical line component:

Definition 2.2.5. f has a vertical line component at α, if

$$\forall \beta \in \mathbb{R} : f(\alpha, \beta) = 0.$$

The next lemma shows that the content of f is the right tool to compute vertical line components.

Lemma 2.2.6. f has a vertical line component at α if and only if α is a root of $\mathrm{cont}(f)$.

Proof. Write f in the following form

$$f = \sum c_i(x) y^i$$

with $c_i(x) \in \mathbb{Z}[x]$.

By definition, α is a vertical line component means that $\forall \beta \in \mathbb{R} : f(\alpha, \beta) = 0$. Since $\mathbb{R}$ is an infinite field, this is equivalent that the polynomial $f(\alpha, y) = 0 \in \mathbb{R}[y]$. In other words, $c_i(\alpha) = 0$ for each polynomial c_i, and therefore, $x - \alpha$ divides each c_i and therefore also divides the content. □

The lemma implies that $f = \mathrm{cont}(f)\mathrm{pp}(f)$ decomposes the curve into the vertical line components and non-vertical components. By dividing out the content, we can work with a version of the curve without vertical lines (this corresponds to the idea of removing vertical line components in the introduction).

In the proof of Lemma 2.2.6, we used the polynomial $f(\alpha, y) \in \mathbb{R}[y]$. As we will use this construction very often, we introduce an abbreviation for it:

Definition 2.2.7. Let $f \in \mathbb{R}[x, y]$ be a curve, and $\alpha \in \mathbb{R}$. We define

$$f_\alpha(y) := f(\alpha, y) \in \mathbb{R}[y]$$

to be the univariate polynomial where α is set for x.

In the remainder of this section, we assume that f is square free and primitive.

2.2.2 Points on algebraic curves

We recall some basic notation from analysis: For a point p on the curve, the *gradient* is defined to be the vector $(D_x f(p), D_y f(p))^T$. If the gradient is not zero, we can define the *tangent* at p to be the line through p and perpendicular to the gradient vector.

Definition 2.2.8. A point p on a curve f is called *singular*, if $D_x f(p) = D_y f(p) = 0$. Non-singular points are called *regular*.

Some examples of singular points are given in Figure 2.2.1.

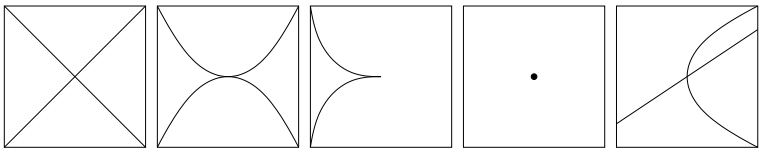

Figure 2.2.1: Some examples of singularities

Regular points have a (unique) tangent line. If this tangent is not vertical, we can parametrise the curve f around p by an *implicit function*. Although the statement is true in more generality, we formulate it only for bivariate polynomials:

Theorem 2.2.9 (Implicit Function Theorem). Let f be a polynomial and $p = (x_0, y_0)$. If $D_y f(p) \neq 0$, there exist open neighbourhoods D around x_0, W around y_0 and a C^∞ function $g : D \to W$ such that, for all $x \in D$, $y = g(x)$ is the unique solution in W for $f(x, y) = 0$. Moreover, the derivative of g is given by:

$$g'(x) = -\frac{D_x f(x, g(x))}{D_y f(x, g(x))}$$

The stated version is stated completely in [Fo05, §8], [Ko93, §3.6]. See also [La68, §XVII.3], [Ap74, §13.4] for slightly weaker formulations.

Geometrically, this means that the curve is given locally around (x_0, y_0) by a function $y = g(x)$.

Example. Consider the unit circle, i.e. the polynomial $f = x^2 + y^2 - 1$. For the upper part of the circle, we can parametrise $g(x) = \sqrt{1 - x^2}$. For the lower part $g(x) = -\sqrt{1 - x^2}$. In either case, all properties of the Implicit Function Theorem are satisfied. Note that the two points where the upper and lower half meet are exactly those points with a vertical tangent line.

There are two classes of points on f that have no such parametrisation: Singular points and points with a vertical tangent line.

Definition 2.2.10. We call a point p *critical*, if $D_y f(p) = 0$.

Theorem 2.2.11. f has only finitely many critical points.

Proof. For a critical point p, we have $f(p) = D_y f(p) = 0$. In other words, p is an intersection point of the curves f and $D_y f$. Since f is square free and primitive, we know from Theorem 2.1.11 that f and $D_y f$ do not share a component. By Bezout's theorem [Wa50, III.3],[BK81, 6.1],[Gi96, 14.4], two algebraic curves without common component intersect in finitely many points. □

To classify critical points, we introduce *x-extreme points*:

Definition 2.2.12. Let $p = (x_0, y_0)$ be a point on f. p is called *x-extreme*, if there exist open neighbourhoods D around x_0, W around y_0 such that x_0 is the greatest (or smallest, respectively) x-coordinate among all points in $D \times W$ on the curve f.

By the Implicit Function Theorem, f can be viewed as a continuous function of x locally around non-critical points. This implies:

Lemma 2.2.13. x-extreme points are critical.

But the converse is not true.

Our definition of x-extremity is analytical, we give an equivalent description in terms of algebraic conditions. Let $D_y^n f$ denote the nth partial derivative of f with respect to y.

Lemma 2.2.14. Let p be a critical regular point and n be the minimal index such that $D_y^n(p) \neq 0$. Then p is *x-extreme* if and only if n is even.

Proof. Let $p = (x_0, y_0)$ be a regular point with vertical tangent line. Since $D_x f(p) \neq 0$, there is a C^∞ function g parametrising f around p, but as a function in the y-coordinate (Implicit Function Theorem). Applying the formula for g' yields:

$$g'(y_0) = -\frac{D_y f(x_0, y_0)}{D_x f(x_0, y_0)} = 0$$

and it follows from one-dimensional analysis that y_0 is an extreme point if and only if the first non-vanishing derivative is even. By using the quotient rule, one can easily show that $g^{(n)}$ is the first non-vanishing derivative. □

By definition, $D_y^n f(\alpha, \beta) = f_\alpha^{(n)}(\beta)$, hence (α, β) is x-extreme if and only if β is a root of f_α with even multiplicity.

Let p be a regular critical point. The previous lemma shows that if it is not x-extreme, it must satisfy $D_y^2(p) = 0$. We define points with this property:

Definition 2.2.15. A point p on the curve f is a *vertical flex* (or *inflection point*), if it is critical, regular and $D_y D_y f(p) = 0$.

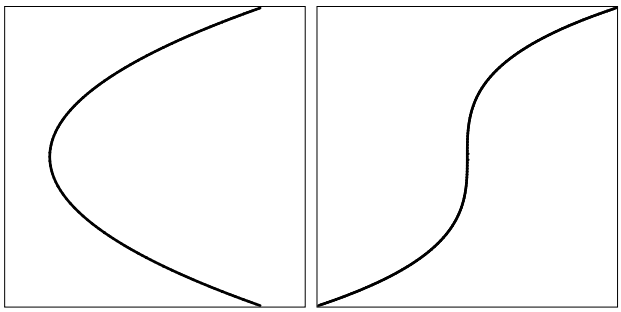

Figure 2.2.2: On the left, an x-extreme point, on the right a vertical flex point.

It follows that each critical point is singular, x-extreme or a flex (but it can both be singular and x-extreme, for instance).

In x-extreme points, the curve changes significantly its behaviour in the following sense: Imagine a vertical line that moves from the left side to the right through the plane. If this line moves over some x-extreme points, the number of intersection points with the algebraic curve changes, whereas this number remains constant for non-x-extreme vertical flexes. This number of intersections might also change at singularities (compare Figure 2.2.1). Therefore, we distinguish a special class of critical points with the next definition.

Definition 2.2.16. A critical point is called *event point* if it is singular or x-extreme.

For the non-event points, we can state a weaker version of the Implicit Function Theorem:

Corollary 2.2.17. For $p = (x_0, y_0)$ a non-event point, there exist open neighbourhoods D around x_0, W around y_0 and a continuous function $g : D \to \mathbb{R}$ such that $f(x, g(x)) = 0$ and $y = g(x)$ is the unique solution in W for $f(x_0, y) = 0$ for all $x \in D$.

Proof. For non-critical points, this follows at once from the Implicit Function Theorem. For non-x-extreme vertical flexes, we first apply the Implicit Function Theorem for the variable y, obtaining a C^∞ function $h(y)$. By definition, we know that y_0 is not a local extremum of h, hence we can deduce that g is strictly monotone in a neighbourhood W of y_0. Then $D := g(W)$ is an open neighbourhood of x_0, and we can define the (continuous, but not differentiable) map $g := h^{-1}$ which has all stated properties. □

2.2.3 Arcs of an algebraic curve

We distinguished the points of our interest in the previous subsection. In the following, we deal with the remaining points of the curve. Intuitively, if we take

out the event points from the curve, the remaining curve decomposes into several connected components. We will call this components *arcs* of the curve. We take a different approach in formally defining these paths, compared to [EK+06] or previous works about EXACUS, but with the same result at the end.

Definition 2.2.18. Let $p = (p_x, p_y), q = (q_x, q_y)$ be two points on the curve f. We say $p \sim q$, if and only if either $p = q$ or there is a continuous map $\phi : [p_x, q_x] \to \mathbb{R}$ (or $[q_x, p_x] \to \mathbb{R}$ respectively) with

1. $\phi(p_x) = p_y$ and $\phi(q_x) = q_y$
2. $\forall x \in [p_x, q_x] : f(x, \phi(x)) = 0$
3. $(x, \phi(x))$ is a non-event point for any $x \in [p_x, q_x]$

A bit more informally, two points on the curve are in relation, if there is a path on the curve connecting these points and not going over any event point of the curve. The following lemma shows that $\sim$ indeed decomposes the curve:

Lemma 2.2.19.

1. For $p \neq q$ and $p \sim q$, the function ϕ is uniquely determined.
2. $\sim$ is a equivalence relation.
3. Every event point is in a singleton equivalence class.

Proof.
1. It is enough to show that for any point $x_0 \in [p_x, q_x]$, ϕ is uniquely determined in an open neighbourhood. This follows from Corollary 2.2.17, since x_0 is a non-event point, and there is only one possible function that parametrises f around x_0.
2. Symmetry and reflexivity hold by construction. For transitivity, assume that $p \sim q$ and $q \sim r$. if $p_x < q_x < r_x$, $p \sim r$ follows by gluing together the functions, otherwise the uniqueness of ϕ yields the equivalence.
3. By definition.

□

Definition 2.2.20. We define an *arc* of f to be the an equivalence class $[p]$ where p is a non-event point. The set of arcs is denoted by $\mathfrak{A}$.

For $A \in \mathfrak{A}$, define the x-range of A as $A_x := p_x(A)$, where p_x is the projection map $(x, y) \mapsto x$.

Lemma 2.2.21. For $A \in \mathfrak{A}$, A_x is an open interval and there is a continuous map $\Phi : A_x \to \mathbb{R}$ such that $A = \{(x, \Phi(x)) \mid x \in A_x\}$.

Proof. For the former statement: A_x is indeed an interval from the definition of $\sim$. Assume that A_x is not open. W.l.o.g., it has a maximum a. There exists a non-event point (a, b) in A by definition. By Corollary 2.2.17, we can find a point (a', b') on f with $a' > a$ and $(a, b) \sim (a', b')$, contradicting the maximality.

For the latter statement, note first that two distinct points with the same x-coordinate are never equivalent under $\sim$. Thus defining for every point $(a, b) \in A$: $\Phi(a) = b$ clearly satisfies the equation above. For any $a \in A_x$, Φ corresponds to the continuous function from Corollary 2.2.17 in an open neighbourhood, so Φ is continuous over A_x. □

We have seen so far: Algebraic curves consist of a finite set of event points and arcs that correspond to graphs of continuous functions on open intervals. What is the limit of that function when x converges to one of its interval boundaries? We distinguish three cases:

Definition 2.2.22. For $A \in \mathfrak{A}$ with parametrisation $\Phi : (a_-, a_+) \to \mathbb{R}$, let

$$p_- := (a_-, \lim_{x \to a_-} \Phi(x)) \in (\mathbb{R} \cup \{\pm\infty\})^2$$

1. If $a_- = -\infty$, the arc is called *arc towards* $-\infty$.

2. If $p_- = (\alpha, \pm\infty)$, the arc is said to *converge to the vertical asymptote* $x = \alpha$ *in direction* $\pm\infty$ *from the right.*

3. If $p_- \in \mathbb{R}^2$, the arc is *incident to* p_- *from the right.*

For $p_+ := (a_+, \lim_{x \to a_+} \Phi(x))$:

1. If $a_+ = \infty$, the arc is called *arc towards* $+\infty$.

2. If $p_+ = (\alpha, \pm\infty)$, the arc is said to *converge to the vertical asymptote* $x = \alpha$ *in direction* $\pm\infty$ *from the left.*

3. If $p_+ \in \mathbb{R}^2$, the arc is *incident to* p_+ *from the left.*

We also call p_-, p_+ *the endpoints* of A.

Not surprisingly, we have the following lemma:

Lemma 2.2.23. If $p \in \mathbb{R}^2$ is an endpoint of an arc A, then p is an event point.

Proof. Let p_i be a sequence of points on A that converges to p. Since f, considered as a polynomial function, is continuous, we have that $f(p) = 0$, so p is on the curve. If it were a non-event point, then p would be on the arc since it is connected by a continuous function. This contradicts the fact that $p \notin A$. □

We see that arcs connect two event points or are unbounded. The definition of incident arcs can be extended to any point p on the curve: We say that an arc A is incident to p from the left and to the right if $p \in A$. For any point p on the curve, we can thus define *the incidence numbers of* p to be the pair (l, r), where l is the number of arcs incident from the left, and r the number of arcs incident from the right. Also, we call the sum of l and r *the total incidence number of* p. We state that any regular point has total incidence number two:

Theorem 2.2.24. Let $p \in \mathbb{R}^2$ be a regular point on the curve f. Then the total incidence number of p is two.

Proof. For non-event points, this follows at once, since they lie on one arc and this arcs counts once at the left and once at the right.

For regular x-extreme points, consider a square centred at p with length a. Now, as a consequence of the Implicit Function Theorem, there is some $a_0 \in \mathbb{R}$ such that the square boundaries intersect the curve exactly twice for any $a < a_0$ and the theorem follows. □

The next criterion characterises regular x-extreme by its incident arcs. We will use this theorem later to detect x-extreme points after a change of coordinates:

Theorem 2.2.25. Let P be a regular point with incident arcs $[p_1]$ and $[p_2]$, where the representatives are non-critical points. Then P is x-extreme if and only if the signs of $D_y f$ at p_1 and p_2 differ.

Proof. First of all, there exist non-critical representatives on the arc, since each arc has infinitely many points, and the number of critical points is finite. All the non-critical points also have the same sign under $D_y f$ since a sign switch would cause an event point on the arc.

W.l.o.g. we can restrict to the case that $P = (\alpha, \beta)$ is critical, otherwise $[p_1] = [p_2]$ and the statement is trivial. By the Implicit Function Theorem, there exist a function $\phi : V \to W$ such that $(\phi(y), y)$ parametrises f around P with

$$\phi'(y) = -\frac{D_y f(\phi(y), y)}{D_x f(\phi(y), y)}.$$

Since $D_x f(P) \neq 0$, we can choose V and W so small that $D_x f$ has constant sign inside $D \times W$. Then, $D_y f$ changes its sign at β if and only if ϕ' changes its sign at β. But the latter happens if and only if P is x-extreme. □

The next question is where vertical asymptotes of f can occur. Consider the easiest example $f = xy - 1$ first. The vertical asymptote is at $x = 0$, and this is a root of the leading coefficient of $f \in (\mathbb{R}[x])[y]$. This holds in general:

Theorem 2.2.26. For $f \in \mathbb{R}[x, y]$, let $r \in \mathbb{R}[x]$ be the leading coefficient of f, considered as a polynomial in y. If $x = \alpha$ is a vertical asymptote of f, then $r(\alpha) = 0$.

Proof. Since $x = \alpha$ is a vertical asymptote, there exists an arc of f converging to $x = \alpha$. Let Φ be the parametrisation of this arc. W.l.o.g., assume that there is a sequence (a_n) with limit α such that $\Phi(a_n)$ converges to $+\infty$.

Assume now that $r(\alpha) \neq 0$. We define $b_n := \Phi(a_n)$ and consider the sequence

$$c_n := \frac{f_{a_n}(b_n)}{f_\alpha(b_n)} = \frac{r(a_n)b_n^k + \dots}{r(\alpha)b_n^k + \dots}$$

It is clear that $c_n = 0$ for all n since the numerator vanishes. On the other hand, polynomial division gives

$$c_n = \frac{r(a_n)}{r(\alpha)} + \frac{\lambda b_n^{k-1} + \dots}{f_\alpha(b_n)}$$

for some $\lambda \in \mathbb{R}[x]$. Since r is continuous, the first quotient converges to one, and the second converges to zero as b_n goes to infinity. So c_n converges to one, contradiction. □

Informally, we can consider $(\alpha, \pm\infty)$ as endpoints of arcs converging to the vertical asymptote $x = \alpha$. In this spirit, we use the following terms:

Definition 2.2.27. Let $f \in \mathbb{R}[x, y]$.

1. $a \in \mathbb{R}$ *supports* a point p, if $f(p) = 0$ and $p = (a, b)$ for some $b \in \mathbb{R}$.
2. $a \in \mathbb{R}$ is a *critical x-value* or *critical x-coordinate*, if a supports a critical point of f, or if $x = a$ is a vertical asymptote of f.
3. $a \in \mathbb{R}$ is an *event x-value* or *event x-coordinate*, if a supports an event point of f, or if $x = a$ is a vertical asymptote of f.

We are now in position to prove:

Theorem 2.2.28. For any curve f, the number of arcs is finite.

Proof. The x-range of any arc is an open interval, and the interval boundaries are either $\pm\infty$, a root of the leading coefficient of f, or the x-coordinate of an event point. So, there are only finitely many such choices. Consequently, if there were infinitely many arcs, there must be some value α that supports infinitely many points of f, contradiction. □

As a corollary, the total incidence number is indeed finite. We remark without proof that it is always an even number, so the number of arcs incident from the left and from the right does not change modulo 2.

Another important property is that the number of points supported by a x-value α remains constant away from event x-values:

Lemma 2.2.29. Let I be an (open or closed) interval such that no event value of f lies inside I. Then, there exists a constant c such that the number of points supported by any $\alpha \in I$ is c.

Proof. Consider two x-coordinates $\alpha_1, \alpha_2 \in I$. Any point p_1 supported by α_1 lies on some arc, and there must be a point p_2 supported by α_2 on the same arc since the arc cannot end between α_1 and α_2. The same holds in the other direction. □

2.3 Real root isolation of polynomials

2.3.1 The Descartes method in the monomial basis

In this section, we discuss the problem of *real root isolation*: Given some polynomial $f \in \mathbb{R}[x, y]$ with n real roots, find *isolating intervals* for each root according to the following definition:

Definition 2.3.1. Let f be a polynomial with root $\alpha \in \mathbb{R}$. A closed interval $[c, d] \subset \mathbb{R}$ containing α and no further root of f is called *isolating interval for α with respect to f* if either $c < \alpha < d$ or $c = \alpha = d$.

For instance, $[1, 2]$ is an isolating interval for $\sqrt{2}$ wrt. x^2-2, for $\sqrt{3}$ wrt. x^3-3x, and also for $\frac{3}{2}$ wrt. $2x-3$, but not for 1 wrt. $x-1$ and not for $\sqrt{2}$ wrt. $(x^2-2)(x^2-3)$.

There are different solutions for the problem of finding isolating intervals for the roots of polynomials, one is based on Sturm's Theorem (treated in Section 2.6). [CL82] is a comprehensive overview over the classical methods. We use an approach based on a well-known property we formulate next. We use the following notation

- Var f := Var$(a_n, \ldots, a_0)$ is the number of sign variation of $f = a_n x^n + \ldots + a_0$. More precisely, when all zeros are deleted from the sequence $(a_n, \ldots, a_0)$, resulting in a new sequence s, then Var$f$ is the number of sign changes in s.
- Let $\alpha_1, \ldots, \alpha_r$ be the positive real roots of f and $s_1, \ldots, s_r$ their multiplicities. We set p_roots$(f) = s_1 + \ldots + s_r$.

Theorem 2.3.2 (Descartes' rule of signs). For any non-zero $f \in \mathbb{R}[x]$, Var$(f)-$ p_roots(f) is non-negative and even.

Proof. Let $f = a_n x^n + \ldots + a_0$. W.l.o.g. we can assume that $f(0) \neq 0$ since we can otherwise divide f by x, only removing a zero root and not changing the sign variation.

To show that Var(f) and p_roots(f) have the same parity, just consider the coefficients a_0 and a_n. The number of sign changes is even if and only if these two coefficients have the same sign. On the other hand, $a_0 = f(0)$, and a_n determines whether $f(x)$ goes to $\pm\infty$ when x goes to $+\infty$. As a continuous function, the number of roots in $(0, +\infty)$, counted with multiplicity is even if and only these two values have the same sign.

To show that Var$f \geq$ p_roots f, we argue by induction over n, the degree of f. If $n = 0$, the statement is obvious. For $n \geq 1$, we assume that the statement is true for the derivative f'. Its coefficients are positive multiples of coefficients of f, so "new" sign variations cannot occur. This means that

$$\text{Var } f \geq \text{Var } f'.$$

For the positive roots of f', we know that there is always a root of the derivative between two distinct roots of f (Rolle's theorem [La68, §III.3],[Fo01, §16]). A k-fold root of f causes a $k-1$-fold root of f'. This gives us

$$\text{p\_roots } f' \geq \text{p\_roots } f - 1$$

By induction hypothesis, $\text{Var} f - \text{p\_roots } f \geq -1$. Since we already know that this difference is always even, the result follows. □

To get an upper bound of the real roots of f in any interval (c, d), one can use the Möbius transformation

$$\phi_{c,d} : (0, \infty) \to (c, d), x \mapsto \frac{cx + d}{x + 1}$$

which defines a bijection. The sign variations of

$$T_{f,c,d}(x) := (x + 1)^{\deg(x)} \cdot (f \circ \phi_{c,d})(x)$$

exceed the number of real roots of f in (c, d) by a non-negative, even integer. Note that the power of $(x + 1)$ is necessary to get a polynomial.

If the number of sign variations is zero or one, the exact number of real roots is known directly according to Theorem 2.3.2. We state an algorithm to isolate the real roots of a polynomial f in an interval (c, d) ([CA76],[CL82],[KM06]):

Algorithm 2.3.3 (Descartes method in power basis).
Input: Interval $[c, d]$, $f = \sum_{i=0}^{n} a_i x^i, f(c) \neq 0 \neq f(d)$
Output: Isolating intervals for the roots of f in (c, d).

1. Compute $v = \text{Var}(T_{f,c,d})$
2. If $v = 0$, return. If $v = 1$, report the interval $[c, d]$ and return.
3. Otherwise, choose a split point $m \in (c, d)$. If $P(m) = 0$, report the interval $[m, m]$. Isolate the roots of f in (c, m) and (m, d).

Two problems still arise here: First of all, the algorithm does not terminate in cases where f has a real multiple root, except if this root is chosen as split point. To overcome this problem, we restrict to square free polynomials (Definition 2.1.9 and Lemma 2.1.20).

A second problem is that even for square free polynomials, it is not obvious that the sign variations eventually drops to zero, if no real root is contained in the interval, and drops to one if there is exactly one root in the interval. But these two properties are needed for termination, and they are implied by the following two theorems:

Theorem 2.3.4 (One-circle theorem). If f has no (complex) roots in the open disk

$$\{z \in \mathbb{C} \mid |z - \frac{c + d}{2}| < \frac{c + d}{2}\}$$

then Var $T_{f,c,d} = 0$.

Theorem 2.3.5 (Two-circle theorem). If f has exactly one root in the union of the two circles

$$\{z \in \mathbb{C} \mid |z - \frac{c+d}{2} \pm i\frac{\sqrt{3}}{6}(d-c)| < \frac{\sqrt{3}}{3}(d-c)\}$$

then Var $T_{f,c,d} = 1$.

Proofs are contained in [KM06]. [ESY06] demonstrates how these bounds can also be used to derive worst-case running times of the root isolation.

As we will also work with non-square-free polynomials later, we will need a generalisation of the previous theorems: For roots with multiplicity k, the sign variation of the isolating interval should be k, if we are close enough at this root. This has been proven recently by Eigenwillig [Eig06]:

Theorem 2.3.6. Let α be a k-fold root of f and let (c, d) be an open isolating interval containing α. If no root of $f^{(k)}$ is inside the disk

$$\{z \in \mathbb{C} \mid |z - \frac{c+d}{2}| < \frac{c+d}{2}\},$$

then Var $T_{f,c,d} = k$.

2.3.2 Upper bounds for roots

Algorithm 2.3.3 gives a solution for the real root isolation in some given interval. To isolate all real roots of f, one must initially enclose them in a sufficiently big interval. We learned about the subsequent theory from the PhD thesis of Batra [Ba99].

The following bound and its proof are taken from [Sl70].

Theorem 2.3.7. For a polynomial $f = a_n x^n + \ldots + a_0$ and any root α of f, it holds that

$$|\alpha| < S(f) := 2\max\left\{\left|\frac{a_{n-1}}{a_n}\right|, \sqrt{\left|\frac{a_{n-2}}{a_n}\right|}, \ldots, \sqrt[n-1]{\left|\frac{a_1}{a_n}\right|}, \sqrt[n]{\left|\frac{a_0}{2a_n}\right|}\right\}$$

Proof. W.l.o.g. we can assume that $\alpha_n = 1$, since scaling does not change the roots of the polynomial. Thus we can rewrite the root bound as

$$S(f) = \max_{i=0,\ldots,n-1} \sqrt[i]{\frac{|a_i|}{\lambda_i}}$$

with $\lambda_i = 2^{-n+i}$ for $i = 1, \ldots, n-1$ and $\lambda_0 = 2^{-n+1}$. In particular,

$$\sum_{i=0}^{n-1} \lambda_i = 1.$$

Therefore, for any $|z| > S(f)$, we can conclude that $|z|^i > \frac{|a_{n-i}|}{\lambda_i}$ and hence

$$\lambda_i > \frac{|a_{n-i}|}{|z|^i}$$

Since the λ_i's sum up to 1, we have

$$1 > \sum_{i=1}^{n} \frac{|a_{n-i}|}{|z|^i}$$

Now, let α be any non-zero root of f. It is enough to show that the sum above is at least one. Indeed,

$$\begin{aligned} |\alpha^n| \left| \sum_{i=1}^{n} \frac{a_{n-i}}{\alpha^i} \right| &= \left| \sum_{i=1}^{n} a_{n-i} \alpha^{n-i} \right| \\ &= \left| \sum_{i=0}^{n-1} a_i \alpha^i \right| \\ &= |\alpha^n| , \end{aligned}$$

and therefore,

$$1 = \left| \sum_{i=1}^{n} \frac{a_{n-i}}{\alpha^i} \right| \leq \sum_{i=1}^{n} \frac{|a_{n-i}|}{|\alpha|^i}.$$

□

The proof shows a bit more: Any choice of positive values λ_i gives a root bound, if $\sum \lambda_i \leq 1$. This general result already appeared in a work of Fujiwara [Fu16], and we therefore call $S(f)$ the *Fujiwara bound* of f. Mignotte and Stefanescu refer to $S(f)$ with the same naming for general λ_i [MS99, §2.5.1].

Call f^γ the polynomial that multiplies all roots of f by γ, in other words $f^\gamma(x) := f(\frac{x}{\gamma})$. For the coefficients $\tilde{a_i}$ of f^γ, we have that $\tilde{a_i} = \frac{a_i}{\gamma^i}$, and from the definition of $S(f)$, it follows that $S(f)$ is scale-independent:

$$S(f^\gamma) = \gamma S(f). \tag{2.3.1}$$

Furthermore, $S(f)$ is a nearly optimal root bound in the following sense:

Theorem 2.3.8. Let $f = a_n x^n + \ldots + a_0$ be a polynomial. The polynomial $|a_n|x^n - |a_{n-1}|x^{n-1} - \ldots - |a_0|$ has a unique positive root α_0. It holds that

$$S(f) \leq 2\alpha_0.$$

Proof. The existence and uniqueness of α_0 follows from Descartes' rule of sign (Theorem 2.3.2).

Because of (2.3.1), it suffices to prove the statement for $\alpha_0 = 1$, and we can w.l.o.g. assume that f is monic. Then, the polynomial

$$x^n - |a_{n-1}|x^{n-1} - \ldots - |a_0|$$

has a root at 1 and therefore, the sum $\sum_{i=0}^{n-1} |a_i| = 1$. It follows that each coefficient of f has absolute value smaller one and consequently, $S(f) \leq 2$. □

The former theorem shows that $S(f)$ is at most twice the optimal root bound of f, if only the absolute values of the coefficients are considered. [Sl70] includes more examples of root bounds and other interesting properties of $S(f)$.

The Fujiwara bound equips us with a complete algorithm for isolating real roots of a square free polynomial f: We first apply the Fujiwara bound to find an initial interval for all real roots of f. Then the Descartes algorithm for square free polynomials 2.3.3 finds isolating intervals for the real roots.

2.3.3 The Descartes method in Bernstein basis

We introduce the *Bernstein polynomials* which will lead to an easier description of the Descartes method and allow to develop a version of the Descartes algorithm that only works with approximate coefficients.

Definition 2.3.9. For $n \in \mathbb{N}$ and $c, d \in \mathbb{R}$ with $c < d$, the *ith Bernstein polynomial of degree n* is defined as

$$B_i^n[c,d](x) := \binom{n}{i} \frac{(x-c)^i (d-x)^{n-i}}{(d-c)^n}$$

for $0 \leq i \leq n$.

The Bernstein polynomials span the vector space of polynomials of degree $\leq n$:

Theorem 2.3.10. $B_0^n[c,d], \ldots, B_n^n[c,d]$ is a basis of the vector space $\mathbb{R}[x]_{\leq n}$.

Proof. Set

$$S := \left\{ \sum_{i=0}^{n} c_i B_i^n[c,d] \mid c_i \in \mathbb{R} \right\}$$

the space spanned by the Bernstein polynomials. It is enough to show that all elements of the basis $(x-c)^0, \ldots, (x-c)^n$ are in S. This is clear for $(x-c)^n = (d-c)^n \cdot B_n^n[c,d]$. Now, for any $k \in \{0, \ldots, n-1\}$, if $(x-c)^m \in S$ for all $m \in \{k+1, \ldots, n\}$, we have that

$$\begin{aligned}
\underbrace{\frac{(d-c)^n}{\binom{n}{k}} \cdot B_k^n[c,d](x)}_{\in S} &:= (x-c)^k (d-x)^{n-k} \\
&= (x-c)^k (d-c-(x-c))^{n-k} \\
&= (d-c)(x-c)^k + \underbrace{\sum_{i=k+1}^{n} \lambda_i (x-c)^i}_{\in S}
\end{aligned}$$

with some λ_i, and therefore, also $(x-c)^k \in S$. □

So we can write each polynomial in $\mathbb{R}[x]_{\leq n}$ as

$$f(x) \quad = \quad \sum_{i=0}^{n} b_i B_i^n[c,d](x) \tag{2.3.2}$$

with some $b_i \in \mathbb{R}$. We call the b_i's *Bernstein coefficients* of f with respect to the interval $[c,d]$.

For shorter notation, we set $B_i := B_i^n[c,d]$. If we apply our transformation function $T_{\cdot,c,d}$ on B_i, a small calculation shows that

$$T_{B_i,c,d} = (x+1)^n \binom{n}{i} \frac{(\phi_{c,d}(x)-c)^i (d-\phi_{c,d}(x))^{n-i}}{(d-c)^n} = \binom{n}{i} x^{n-i}.$$

Since $T_{\cdot,c,d}$ is a linear mapping, It follows with (2.3.2):

$$T_{f,c,d} = \sum_{i=0}^{n} b_i T_{B_i,c,d} = \sum_{i=0}^{n} \binom{n}{i} b_i x^{n-i}.$$

This gives a reformulation of Descartes' rule:

Theorem 2.3.11 (Descartes' rule of sign in the Bernstein basis). For nonzero $f = \sum_{i=0}^{n} b_i B_i^n[c,d]$, let v be the number of sign variations of the Bernstein coefficients and let r be the number of real roots in (c,d), counted with multiplicities. Then $v-r$ is non-negative and even.

With Theorem 2.3.11, the root isolation simplifies when the Bernstein basis is used consequently:

Algorithm 2.3.12 (Isolation of real roots in Bernstein basis).
Input: Interval (c,d), $f = \sum_{i=0}^{n} b_i B_i^n[c,d]$ square free, $f(c) \neq 0 \neq f(d)$.
Output: Isolating intervals of the roots of P in (c,d).

1. Compute v, the number of sign variations of the Bernstein coefficients.
2. If $v = 0$, return. If $v = 1$, report the interval $[c,d]$ and return.
3. Otherwise, choose a rational split point $m \in (c,d)$. If $f(m) = 0$, report the interval $[m,m]$.
4. Compute the basis representation with respect to the Bernstein polynomials $(B_i^n[c,m])_{i=0}^n$ and $(B_i^n[m,d])_{i=0}^n$.
5. Isolate the roots of f in (c,m) and (m,d) recursively.

A straightforward and efficient solution for step 4 is the famous *de Casteljau algorithm*. Its idea is to start with the sequence $(b_0, \ldots, b_n)$ and to form a triangular shape of numbers by iteratively adding neighbours. The edges of this triangle give the Bernstein coefficients with respect to $[c, m]$ and $[m, d]$. See [BPR03, Algorithm 10.2], or [ESY06] and the references therein for more detailed treatments.

The recursion steps in the Descartes method (in the power basis and in Bernstein basis) can be represented in a binary tree structure: Every node corresponds to one visited interval and one node v_1 is an ancestor of v_2 if the interval of v_2 is a subset of v_1.

We will mark every node with the sign variation of the corresponding interval. It is then clear that nodes marked with 0 or 1 are leafs in the tree. If we assume that no roots occur on the chosen split points, the leaves are exactly the nodes marked with 0 and 1. Figure 2.3.1 shows one example.

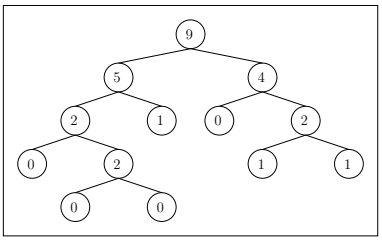

Figure 2.3.1: A possible Descartes tree. Note that the sum of the children is not necessarily equal to the parents sign variation, but they differ by an even number (assuming that no split point is a root)

The following observation is crucial for our applications. It says that the mark of a node is always at least the sum of its children, and both differ by an even number.

Lemma 2.3.13. Let c, d, m be non-roots of f and v, v_1, v_2 be the number of sign variations of the Bernstein coefficients of f with respect to $[c, d]$, $[c, m]$ and $[m, d]$, respectively. Then $v - (v_1 + v_2)$ is non-negative and even.

This follows quite directly by considering the de Casteljau triangle. See [BPR03, Prop.10.34] for a proof.

We explain next that it is possible to transform Algorithm 2.3.12 such that only approximations of the coefficients are necessary: Since the real roots of a polynomial depend continuously on the coefficients, isolating intervals remain isolating if the polynomial's coefficients are slightly perturbed (or closely approximated). Thus, as long as we can be sure to take the correct decision in each step of Algorithm 2.3.12, we can just use approximations of the Bernstein coefficients.

Eigenwillig et al. ([EK+05]) describe how this idea can be used to build the so-called *Bitstream Descartes method*. There are two main obstacles: First, there is a

problem if a chosen split point is a root of the polynomial, since this is impossible to test by only calculating approximate. This problem is solved via randomised choices of the split points during the algorithm.[1]

Second, it can happen that the sign of some coefficient during the computation cannot be determined. So, there is not a unique number of sign variation of an interval, but a set of possible numbers. At least, the algorithm ensures (again by the randomised choice of split points) that the sign of the first and last Bernstein coefficient (which correspond to the function values at the interval endpoints) are determined, hence the parity of each number of sign variations is known. The authors show that isolating intervals for simple roots of the exact polynomial and intervals without any root can be determined if the coefficients are sufficiently approximated. Of course, it is not known a priori how much the coefficients must be approximated for a successful run of the algorithm, and the precision must be increased in situations where the algorithm fails to find suitable split points. For a detailed treatment, see [EK+05].

To illustrate why the Bitstream Descartes algorithm is very useful for our purposes, consider a square free bivariate polynomial $f \in \mathbb{Z}[x, y]$. We want to find the roots of the polynomial $f_\alpha = f(\alpha, y) \in \mathbb{R}[y]$ for some $\alpha \in \mathbb{R}$ (the roots correspond to the y-coordinates of points on the curve with x-coordinate α). The coefficients of f_α are not integers in general, but coefficient over the domain $\mathbb{Z}[\alpha]$, and performing a usual Descartes algorithm would result in expensive algebraic calculation with α. However, the Bitstream Descartes method allows us to find isolating intervals only by computing approximations of the coefficients. How to find these approximations for real algebraic numbers is the subject of Section 2.4.

2.3.4 Root isolation for polynomials with multiple roots

For the Descartes method (in Bernstein basis or power basis), we had to restrict to square free input polynomials. That's a serious restriction because polynomials with multiple root will appear all over the place in our algorithm. The simplest solution to overcome this problem is the following lemma. We only show it for real polynomials although it remains valid for arbitrary factorial domains:

Lemma 2.3.14. Let f be a polynomial in $\mathbb{R}[x]$. Then, $\frac{f}{\gcd(f,f')}$ is square free and has the same roots as f.

Proof. W.l.o.g. we assume that f is monic. We can decompose:

$$f = (x - \alpha_1)^{e_1} \cdot \ldots \cdot \ldots (x - \alpha_r)^{e_r} = (x - \alpha_1)^{e_1} \cdot F$$

with distinct $\alpha_i \in \mathbb{C}$. Recall that by definition, e_i gives the multiplicity of the root α_i.

[1] In a second variant, the problem is solved via performing a random shift of the input coefficient and choosing always the midpoint of the interval.

Computing the derivative, we see that

$$f' = (x - a_1)^{e_1} \cdot F' + e_1(x - a_1)^{e_1 - 1} \cdot F$$

We see that α_1 is a root of the gcd of f and f' with multiplicity $e_1 - 1$. In particular, if α_1 is a simple root of f, it is not a root of f'. The same holds for each α_i. Therefore,

$$\gcd(f, f') = (x - \alpha_1)^{e_1 - 1} \cdot \ldots \cdot \ldots \cdot (x - \alpha_r)^{e_r - 1}$$

and the claim follows. □

Consequently, a solution for non-square-free polynomials is to divide out the greatest common divisor and apply the Descartes method on the square free version of the polynomial. We remark that this technique is not always a practicable way of isolating roots: If the coefficients of the polynomial are not integers, but from the domain $\mathbb{Z}[\alpha]$ with α algebraic, the gcd-computation becomes extremely complicated. A main result of this work is to give alternative solution for the isolation of multiple roots based on the Descartes method, exploiting additional properties of the polynomial.

2.4 Representation of real algebraic numbers

We introduce the interval representation of general real algebraic numbers and present some basic operations. For approximations, we need some properties of *interval arithmetic* which we discuss first. A detailed introduction to interval arithmetic can be found in [Mo79].

Interval arithmetic

Assume that a number $\alpha \in \mathbb{R}$ is not known exactly, but it is assured that it is contained in some interval I. For an operation op on real numbers, we want to define an operation $\overline{\text{op}}$ on intervals such that $\text{op}(\alpha) \in \overline{\text{op}}(I)$. For basic operations like addition and multiplication, this is straightforward to define:

Definition 2.4.1. For intervals $[a, b]$, $[c, d]$ and $\lambda \in \mathbb{R}$, we define

1. $\lambda + [a, b] := [\lambda + a, \lambda + b]$,
2. $[a, b] + [c, d] := [a + c, b + d]$,
3. $\lambda \cdot [a, b] := \begin{cases} [\lambda \cdot a, \lambda \cdot b] & \text{if } \lambda \geq 0, \\ [\lambda \cdot b, \lambda \cdot a] & \text{if } \lambda < 0, \end{cases}$
4. $[a, b] \cdot [c, d] := [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}]$.

For an interval I and $f = \sum_{i=0}^{n} a_i x^i \in \mathbb{R}[x]$, we evaluate f at the interval I according to the *Horner scheme* and write:

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

For $\alpha \in I, \beta \in J$, we have $-\alpha \in -I$, $\alpha + \beta \in I + J$, $\alpha \cdot \beta \in I \cdot J$ etc. Thus it also follows that $f(\alpha) \in f(I)$. We show next that $f(\alpha)$ can be approximated up to arbitrary precision by refining the interval of α. Therefore, we use the following famous theorem about nested intervals [Fo01, §5]. For $I = [c, d]$, define the length of I as $\mathfrak{L}(I) := d - c$.

Theorem 2.4.2. Let (I_k) be a sequence of intervals with $I_0 \supset I_1 \supset \ldots$ such that $\lim_{k\to\infty} \mathfrak{L}(I_k) = 0$. Then, there exists an $\alpha \in \mathbb{R}$ such that

$$\bigcap_{i=0}^{\infty} I_k = \{\alpha\}$$

We call (I_k) a *nested interval sequence* for α.

Proof. If the intersection of all intervals contained more than one point, all intervals would contain two points with a fixed distance δ and so $\mathfrak{L}(I_k) \geq \delta$, contradicting the assumption. It is therefore enough to show that the intersection is not empty.

We define $[a_k, b_k] := I_k$. The sequence (a_k) is a Cauchy-sequence. By the completeness axiom, (a_k) converges to some $\alpha \in \mathbb{R}$. Since the a_k are monotone, it holds that $a_i \leq \alpha$. Furthermore, each b_j is an upper bound for the sequence (a_i) and therefore $\alpha \leq b_j$. In other words, $\alpha \in I_k$ for all $k \in \mathbb{N}$. □

Lemma 2.4.3. Let $\lambda \in \mathbb{R}$ and $(I_k), (J_k)$ nested interval sequences for α and β. Then

1. $(\lambda + I_k)$ is a nested interval sequence for $\lambda + \alpha$.
2. $(I_k + J_k)$ is a nested interval sequence for $\alpha + \beta$.
3. $(\lambda \cdot I_k)$ is a nested interval sequence for $\lambda \cdot \alpha$.
4. $(I_k \cdot J_k)$ is a nested interval sequence for $\alpha \cdot \beta$.

Proof. We only prove the last statement, the others are easy.

First of all, it holds that $I_0 \cdot J_0 \supset I_1 \cdot J_1 \supset \ldots$. Next, we show that the length indeed converges to zero: There exist numbers $a_k, A_k \in I_k, b_k, B_k \in J_k$ such that

$$I_k \cdot J_k = [a_k \cdot b_k, A_k \cdot B_k].$$

As a consequence of Theorem 2.4.2, (a_k) and (A_k) converge to α, and (b_k) and (B_k) converge to β. So $(a_k \cdot b_k)$ converges to $\alpha \cdot \beta$ as well as $(A_k \cdot B_k)$. □

Since polynomial evaluation is a finite sequence of additions and multiplications, this lemma implies

Theorem 2.4.4. If (I_k) is a nested interval sequences for α, $(f(I_k))$ is a nested interval sequence for $f(\alpha)$.

Exact representation

Recall the definition of algebraic from Section 2.1.3: α is algebraic over a domain D, if there exists a non-zero polynomial $f \in D[x]$ such that $f(\alpha) = 0$. From now, we will call $c \in \mathbb{C}$ *algebraic*, if it is algebraic over $\mathbb{Z}$. We will call it *real algebraic* if it is also a real number. As already remarked in Section 2.1.3, being algebraic over $\mathbb{Z}$ and being algebraic over $\mathbb{Q}$ is equivalent.

For instance, $\sqrt{2}$ is algebraic since it is a root of $x^2 - 2$. But root expressions are not a general representation for (real) algebraic numbers – the famous Galois theory (covered by most textbooks in Algebra, for instance [La93, Bo01, Wo96, Wa71]) shows that not all polynomial equations are solvable with radicals. Instead, we represent real algebraic numbers in the following way:

Definition 2.4.5. Let α be a real algebraic number that is a root of $f \in \mathbb{Z}[x]$. If $I = [c, d]$ is an isolating interval of f for α, we call (f, I) an *(integral) interval representation* of α. We also write $\alpha \mathrel{\widehat{=}} (f, I)$. We call an interval representation *simple*, if α is a simple root of f, i.e. a root of multiplicity one.

The algebraic number α is uniquely identified by an interval representation, although neither f nor I are unique. For instance, $\sqrt{2}$ has the representation $(x^2 + 2, [0, 2])$, but also $(x^3 + 2x, [1, 4])$.
The results from the previous section can be used to compute a simple interval representation of any real root of f. The naive approach would be to divide out $\gcd(f, f')$ first to make the polynomial square free (compare Lemma 2.3.14), and then isolate the roots using the Descartes method. However, for computational purposes, it is an advantage if the defining polynomial for the roots are of smaller degree. So, we factorise the square free part of f into polynomials $F_1, \ldots, F_r$, such that F_i contains exactly the roots of f with multiplicity i. We refer to [GCL92, §8.1] for an excellent and much more detailed treatment:

Algorithm 2.4.6 (Square free factorisation).
Input: Polynomial $f \in \mathbb{Z}[x]$.
Output: Polynomials $F_1, \ldots, F_r$ such that

$$f = \prod_{i=1}^{r} F_i^i$$

1. Set $g \leftarrow \gcd(f, f')$, $f^* \leftarrow \frac{f}{g}$, $i \leftarrow 1$
2. While $g \neq 1$, do the following:
   i. $y \leftarrow \gcd(g, f^*)$
   ii. Set $F_i \leftarrow \frac{f^*}{y}$, $i \leftarrow i + 1$.
   iii. Update $f^* \leftarrow y$ and $g \leftarrow \frac{g}{y}$
3. Set $F_i \leftarrow f^*$

For our implementation, we use a slight modification of that algorithm, called *Yun's Square-Free Factorisation*. See [GCL92, Algorithm 8.2] and [Yun76] for details.

We formulate an algorithm to create interval representations of algebraic numbers:

Algorithm 2.4.7.
Input: An integer polynomial $f \in \mathbb{Z}[x]$
Output: A sequence of simple interval representation (f, I)

1. Decompose $f = F_1, \ldots, F_r$ as above.

2. For each $i = 1, \ldots, r$, apply the Descartes algorithm to find isolating intervals.

3. Output the representations

We remark that the output list is not ordered so far. As we will see later in this section, it is easily possible to compare two algebraic numbers in interval representation. Hence, the algorithm above can be easily modified to produce a increasing list of algebraic numbers.

We now study what operations we can perform efficiently with our representation of algebraic numbers. First, we demonstrate how the isolating interval can be shrunk to arbitrary width, leading to a numeric approximation of the number:

If $\alpha = (f, (c, d))$ is simple, it holds that $\operatorname{sgn}(f(c)) \neq \operatorname{sgn}(f(d))$. To make the interval smaller, one can just use the well-known *bisection*: Choose a split point (e.g. the midpoint $m = \frac{c+d}{2}$), and look for the subinterval with a sign change at the boundaries. However, each step only gives us one more bit precision. Another well-known method is the *Newton method*: It converges quadratically in good cases, but it can diverge, too. Also the denominators appearing in the iteration can become complicated (whereas in the bisection, we can always chose split points such that the denominator is a power of 2). Abbott ([Ab06]) describes the *quadratic interval refinement* method, a combination of bisection and *regula falsi*. Its advantage is that it always terminates and eventually doubles the number of exact bits in every step. We give a short description of the nice idea:

Let N be an integer initially set to 4. We divide the isolating interval (c, d) of α into N subintervals, with $N + 1$ equidistant boundary points. Now, we construct the line through the points $(c, f(c))$ and $(d, f(d))$. Since the signs of $f(c)$ and $f(d)$ differ, this line intersects the x-axis between c and d. Let m be the boundary point nearest to that intersection point. If the signs of $f(c)$ and $f(m)$ differ, we expect the root to be in the subintervals with m as right boundary, otherwise we expect it to be in the subinterval with m as left boundary. We call this interval J.

There are two cases: If $N = 4$, two bisection steps are performed and it is checked whether the resulting isolating interval equals J. If it does, N is set to $N^2 = 16$. Otherwise, N remains 4.

If $N \neq 4$, it is simply checked whether the signs at the boundaries of J differ. If so, then J is taken as new isolating interval, and N is set to N^2. If there is no sign change, the isolating interval remains unchanged, and N is set to $\sqrt{N}$.

The method terminates when the isolating interval is small enough. This happens eventually, since there is always a refinement if $N = 4$, and otherwise N is decreased if no refinement takes place.

The idea is also illustrated in Figure 2.4.1. Intuitively, the function is better and better approximated by the resulting secants, and the guessed interval J will be eventually always right and doubles the precision in each step. [Ab06] also contains a proof for the quadratic convergence – we will focus on the practical behaviour of the method compared to simple bisection in the experiments in Chapter 6.

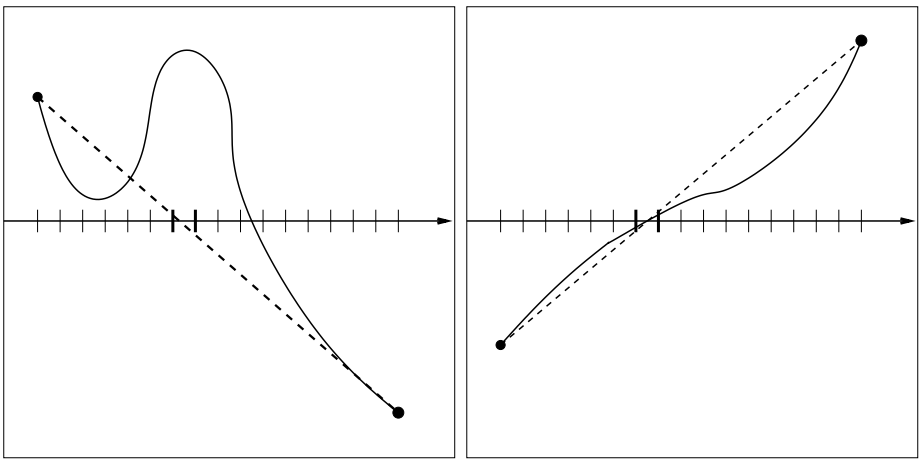

Figure 2.4.1: In the left picture, the guessed interval does not contain the real root, therefore, N is decreased to 4 in the next step. On the right, there is a sign change in the guessed interval. Therefore, it is taken as refined isolating interval and N is increased to 256.

Knowing how isolating intervals can be refined, we can now compare two real algebraic numbers in simple interval representation. For the equality test, we need symbolic computations:

Algorithm 2.4.8 (Equality).
Input: Two simple representations $\alpha \mathrel{\widehat{=}} (f, I)$ and $\beta \mathrel{\widehat{=}} (g, J)$.
Output: true iff $\alpha = \beta$

1. If I and J are disjoint intervals, return false

2. Set $K := I \cap J$. If K is not isolating for α as well as for β, return false. This can be checked by comparing the signs of f and g at the boundaries of K.

3. Compute $h := \gcd(f, g)$. Return whether h switches sign at the boundaries of K.

If α and β are not equal, one can refine the isolating intervals of both until they are disjoint and deduce whether α is greater or less than β.
Next, we consider the problem of evaluating polynomials in real algebraic numbers.

Algorithm 2.4.9 (zero test for g at α).
Input: $g \in \mathbb{Z}[x]$, $\alpha = (f, I)$.
Output: True if and only if $g(\alpha) = 0$

1. Compute $h := \gcd(f, g)$
2. Check whether the signs of h at the boundaries of I differ.

The correctness is immediate: $g(\alpha) = 0$ if and only if $h(\alpha) = 0$. Since the roots of h form a subset of the roots of f, I must also be an isolating interval for h in this case.
If $g(\alpha) \neq 0$, we compute the sign of $g(\alpha)$ as follows:

Algorithm 2.4.10 (non-zero sign of g at α).
Input: $g \in \mathbb{Z}[x]$, $\alpha = (f, I)$ with $g(\alpha) \neq 0$.
Output: The sign of $g(\alpha)$

1. Evaluate $J := g(I)$, using interval arithmetic.
2. If both interval boundaries have the same sign, return it.
3. Otherwise, refine the interval I to I'
4. Call the algorithm recursively with the representation $\alpha = (f, I')$

The algorithm terminates since the isolating intervals used form a nested interval sequence for α, so the interval boundaries of the J's converge to $g(\alpha)$. Since this is non-zero, both boundaries have eventually the same sign.

For a numerical approximation of $f(\alpha)$, we use the same principle. The termination criterion is that the length of $g(I)$ becomes smaller than some given ϵ.

We want to derive a second, more general representation of real algebraic numbers. First of all, we recall from Section 2.1.3 that $\overline{\mathbb{Q}}$, the algebraic closure of $\mathbb{Q}$, is an algebraically closed field. Moreover, the extension $\mathbb{Q} \subset \overline{\mathbb{Q}}$ is algebraic, this means that $\overline{\mathbb{Q}}$ only contains algebraic elements. This shows that $\overline{\mathbb{Q}}$ is precisely the set of algebraic numbers. As a consequence, the roots of any $g \in \overline{\mathbb{Q}}[x]$ are algebraic numbers again. Such roots arise, for instance, when a bivariate polynomial $f \in \mathbb{Z}[x, y]$ is evaluated at the position $x = \alpha$ with $\alpha \in \overline{\mathbb{Q}}$. We give such (real) roots a special representation:

Definition 2.4.11. Let $f \in \mathbb{Z}[x, y]$, $\alpha, \beta \in \mathbb{R}$ and α be in interval representation. If $f(\alpha, \beta) = 0$ and I is an isolating interval of f_α for β with rational boundaries, we call (f, α, I) an *algebraic interval representation* of β and write $\beta \mathrel{\widehat{=}} (f, \alpha, I)$. It is called *simple* if β is a simple root of f_α.

In other words (f, α, I) is the representation of a root of f_α. It is still possible to refine I by bisection, if the representation is simple: All we need are the signs of f_α, evaluated at the boundaries of I, and at the split point. But this is nothing but a sign computation of the polynomial $f(x, q)$ (where q is one of these three rational values) at the real algebraic number α. We will see later that it is also possible to refine the isolating interval in non-simple cases.

Using that refinement property, it is also possible to compare two different algebraic interval representations by merely refining their isolated intervals until they are disjoint.

As a conclusion, here are the operations that we have described for our representations of real algebraic numbers. For simple integer interval representations $\alpha \mathrel{\widehat{=}} (f, I)$:

- Refine I to any precision
- Compare α to another real algebraic number of that kind
- Compute the sign of $g(\alpha)$ for a polynomial $g \in \mathbb{R}[x]$
- Compute $g(\alpha)$ up to any precision

For simple algebraic interval representations $\beta \mathrel{\widehat{=}} (f, \alpha, I)$:

- Refine I to any precision
- Compare β to any different real algebraic number

We remark that there exist methods to transform algebraic numbers from the algebraic interval representation into integral interval representation [Lo82b]. We abstain from using these methods for efficiency.

2.5 Subresultants and the greatest common divisor

We give a geometric motivation: Imagine two curve f, g in the plane and some x-coordinate α. We are interested in intersection points of these curve at α as a geometric property. To find them, one has to compute the greatest common divisor of $f_\alpha = f(\alpha, y)$ and $g_\alpha = g(\alpha, y)$. Instead of calculating this gcd with the Euclidean algorithm, one can use *subresultants*, a sequence of polynomials. Their most important properties are

- They are defined as algebraic expressions in the coefficients of f and g.
- The formal leading coefficients of the subresultants provide the degree of the greatest common divisor.
- The sequence of subresultants contains all polynomials occurring in the Euclidean algorithm (in particular the gcd itself), up to a scalar factor. In fact, their size is smaller than the size of the Euclidean polynomials.

- If the polynomials have only one common root, one can express this root as rational expression in the subresultant coefficients.
- The subresultant are well-behaved under homomorphisms: Suppose ϕ is a homomorphism of domains from D to D' which preserves the degree of f and $g \in D$. One can obtain a subresultant sequence for $\phi(f)$ and $\phi(g)$ by applying ϕ on the subresultant sequence of f and g.

Especially the last property is extremely useful in geometric applications as we show at the end of this section. We discuss the named properties one after another. Throughout this section, let D be a factorial domain and

$$f = \sum_{i=0}^{n} f_i x^i, \qquad g = \sum_{i=0}^{m} g_i x^i$$

polynomials in $D[x]$ with $n = \deg(f) \geq \deg(g) = m$.

The *kth Sylvester submatrix* of f and g is defined to be the following $(n+m-2k) \times (n+m-k)$ matrix, composed of $m-k$ rows of coefficients of f and $n-k$ rows of coefficients of g:

$$\mathrm{Syl}_l(f,g) = \begin{pmatrix} f_n & & \cdots & f_0 & & \\ & \ddots & & & \ddots & \\ & & f_n & \cdots & & f_0 \\ g_m & & \cdots & g_0 & & \\ & \ddots & & & \ddots & \\ & & g_m & \cdots & & g_0 \end{pmatrix}$$

The matrix $\mathrm{Syl}_0(f,g)$ is also called the *Sylvester matrix*.

The Sylvester submatrices arise naturally by asking if there exist (non-zero) polynomials u, v with $\deg(u) < m-k$, $\deg(v) < n-k$ such that $uf + vg = 0$. Expressed in terms of linear algebra, this corresponds to the linear system

$$(u,v) \cdot \mathrm{Syl}_k(f,g) = 0 \tag{2.5.1}$$

where u and v are identified with their coefficient vector.

Definition 2.5.1. For a polynomial $f = \sum_{k=0}^{n} a_k x^k$, we set

$$\mathrm{coef}_k(f) := a_k.$$

Definition 2.5.2. For $0 \leq k \leq n$, the *kth subresultant of f and g* is defined as

$$\mathrm{Sres}_k(f,g) = \begin{cases} \sum_{i=0}^{k} M_i^k(f,g) x^i & \text{if } k \leq m-1, \\ g & \text{if } k = m, \\ 0 & \text{if } m+1 \leq k < n, \\ f & \text{if } k = n, \end{cases}$$

where $M_i^k(f,g)$ is the determinant of the matrix built with the first $n+m-2k-1$ and the $(n+m-k-i)$th column of the kth Sylvester matrix. The *kth principal subresultant coefficient (psc)* for $k=0,\ldots,n$ is defined as

$$\operatorname{sres}_k(f,g) = \begin{cases} 1 & \text{if } k=n, \\ \operatorname{coef}_k(\operatorname{Sres}_k(f,g)) & \text{if } k=0,\ldots,n-1. \end{cases}$$

We call $\operatorname{coef}_{k-1}(\operatorname{Sres}(f,g)) =: \operatorname{cosres}_k(f,g)$ the *kth coprincipal subresultant coefficient* for $k=1,\ldots,n$.

The sequence $\operatorname{Sres}_n(f,g),\ldots,\operatorname{Sres}_0(f,g)$ is called the *subresultant sequence* of f and g. The sequences of principal and coprincipal coefficients are defined in the same way.

In particular, $\operatorname{Sres}_0(f,g) = \operatorname{sres}_0(f,g) = \operatorname{res}(f,g)$, where $\operatorname{res}(f,g)$ denotes the resultant of f and g.

Some authors call the psc's "subresultants" (or *scalar subresultants*) which might confuse the reader. [GL03] contains an overview how the term "subresultants" is used in the literature.

The psc's provide the degree of the greatest common divisor as we will show next. We use the following lemma that gives another description for the degree of the greatest common divisor [Wol02]:

Lemma 2.5.3. Let f,g be polynomials in $D[x]$ and $k := \deg(\gcd(f,g))$. Then k is the minimal integer i such that all non-zero $u,v \in D[x]$ with $\deg u < m-i$, $\deg v < n-i$ satisfy $\deg(uf+vg) \geq i$.

Proof. Let $h := \gcd(f,g)$, and k its degree. Set $u' := \frac{g}{h}$ and $v' := -\frac{f}{h}$. Clearly, for any integer $i<k$, $\deg(u') = m-k < m-i$ and $\deg(v') = n-k < n-i$, and $u'f+vg' = 0$. This shows the minimal integer with the specified property is at least k.

To show that $i=k$ satisfies the specified property, assume for a contradiction that there exist non-zero polynomials u,v with $\deg(u) < m-k$ and v with $\deg(v) < n-k$ such that $\deg(uf+vg) < k$. Since h divides $uf+vg$, this means that $uf+vg=0$, and also $u\frac{f}{h} + v\frac{g}{h} = 0$. This implies that $\frac{f}{h}$ divides v because $\frac{f}{h}$ and $\frac{g}{h}$ are coprime. But $\deg\frac{f}{h} = n-k > \deg(v)$, a contradiction. □

If one truncates the last i columns of Syl_i in (2.5.1), the new system has a (non-trivial) solution iff there exist polynomials u,v with $\deg(u) < m-i$, $\deg(v) < n-i$, such that $\deg(uf+vg) < i$. By definition, such a solution exists iff $\operatorname{sres}_i(f,g)$, the determinant of the truncated matrix, vanishes. Together with Lemma 2.5.3, this proves that the principal subresultant coefficients are an indicator for the degree of the greatest common divisor:

Corollary 2.5.4.

$$\deg(\gcd(f,g)) \quad = \quad \min\{k \in \{0,\ldots,n\} \mid \operatorname{sres}_k(f,g) \neq 0\}$$

We show next that even more is true: The first non-vanishing subresultant is an associate of the greatest common divisor of f and g. We follow the argumentation from [BT71] and express the ith subresultant as one single determinant.

$$\mathrm{Sres}_i(f,g) \;=\; \det \begin{pmatrix} f_n & \cdots & \cdots & f_{2i-m+2} & x^{m-i-1}f \\ & \ddots & & \vdots & \vdots \\ & f_n & \cdots & f_{i+1} & f \\ g_m & \cdots & \cdots & g_{2i-n+2} & x^{n-i-1}g \\ & \ddots & & \vdots & \vdots \\ & g_m & \cdots & g_{i+1} & g \end{pmatrix} \tag{2.5.2}$$

with $f_j = 0 = g_j$ for negative indices j. The equality follows directly from the linearity of the determinant in the last column.

Laplace expansion in the last column in (2.5.2) yields

$$\mathrm{Sres}_i(f,g) = uf + vg, \quad \deg(u) < m-i, \quad \deg(v) < n-i. \tag{2.5.3}$$

Let $k := \deg(\gcd(f,g))$. By Corollary 2.5.2, $\mathrm{Sres}_k(f,g)$ is a polynomial of degree k, since it has $\mathrm{sres}_k(f,g) \neq 0$ as leading coefficient. With (2.5.3), it follows at once that

$$\mathrm{Sres}_k(f,g) \sim \gcd(f,g). \tag{2.5.4}$$

Even better, the subresultant sequence does not only contain the greatest common divisor, but all polynomials occurring in the Euclidean algorithm, up to associates. We call a subresultant $\mathrm{Sres}_i(f,g)$ *regular*, if $\deg(\mathrm{Sres}_i(f,g)) = i$ and *defective* otherwise. Note that the ith subresultant is defective if and only if $\mathrm{sres}_i(f,g)$ vanishes.

Theorem 2.5.5. Let $S_k, \ldots, S_0$ be the sequence of regular subresultants (starting with $S_k = f, S_{k-1} = g$). Then, there exist $\alpha_i, \beta_i \in D, Q_i \in D[x]$ such that

$$\alpha_i S_{i+1} = Q_i S_i + \beta_i S_{i-1}$$

for $i \in \{1, \ldots, k-1\}$.

Proofs can be found in [Co67, BT71, Lo82a, GL03].

It is possible to calculate the values of α_i, β_i, and this leads to efficient algorithms to compute the subresultant. These expressions are quite complicated in general, but if all subresultants are regular, we have the equality

$$\mathrm{sres}_i^2 \mathrm{Sres}_{i+1} \;=\; Q_i \cdot \mathrm{Sres}_i + \mathrm{sres}_{i+1}^2 \mathrm{Sres}_{i-1} \tag{2.5.5}$$

for some $Q_i \in D[x]$ and all $i = 1, \ldots, n-1$ (compare [Lo82a], [GL03]). In particular, all α_i, β_i are positive.

Consider $C := \overline{Q(D)}$, the algebraic closure of D's quotient field (you can also set $D := \mathbb{Q}$ and $C := \mathbb{C}$ for simplicity). Suppose that f and g only have one

common root over C with multiplicity k. In other words, $\gcd(f,g) = (x-\beta)^k$. Using (2.5.4) gives

$$\mathrm{Sres}_k(f,g) = \mathrm{sres}_k(f,g)(x-\beta)^k,$$

and by comparing the coefficients of x^{k-1}, it follows

$$\beta \;=\; -\frac{\mathrm{cosres}_k(f,g)}{k \cdot \mathrm{sres}_k(f,g)}. \tag{2.5.6}$$

This expression also appears in [GN02]. We see that the multiple root can be expressed as rational expression which depends on the principal and coprincipal subresultant coefficients of f and g.

Let $\varphi : D \to D'$ be a homomorphism of domains such that the leading coefficients of f and g are not in the kernel of φ. This means that φ preserves the degree of f and g, and therefore for any i,

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

where $\varphi(A)$ means applying φ on each entry of the matrix A. Since the determinant is nothing but a sum of products, we have that

$$\varphi(\det A) = \det(\varphi(A)).$$

This proves the following theorem (see also [Yap00, §4.4, Lemma 4.9]):

Theorem 2.5.6 (Specialisation property). Let $\varphi : D \to D'$ be a homomorphism of domains with $\mathrm{lc}(f), \mathrm{lc}(g) \notin \ker\varphi$. Then, for any $i \in \{0,\ldots,n\}$

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

The name of the theorem comes from its main application: Assume $D = R[x]$ is a domain with parameter, and consider the homomorphism to R that specialises x to α. Then, by knowing the subresultant sequence of $f, g \in D$, one obtains the subresultant sequence of f_α, g_α by evaluating $\mathrm{Sres}_i(f,g)$ at $x = \alpha$, in other words

$$\mathrm{Sres}_i(f_a, g_a) = \mathrm{Sres}_i(f,g)(\alpha)$$

Of course, the specialisation works as well in more generality for an arbitrary number of parameters where all or only a part of them are specialised by the homomorphism.

2.6 Sturm-Habicht sequences and real root counting

The subresultant sequence provides information for the intersections of two different polynomials f and g. For our problem of analysing a single curve, we are mainly interested in the special case $g = f'$, as intersections of f and $D_y f$ are critical points of the curve. Apart from the properties derived in the previous section, we can also slightly modify the subresultant sequence of f and f' to count the number of real roots of f over any x-coordinate α. Therefore, we introduce *Sturm-Habicht sequences* and exploit their relation to *Sturm sequences*.

Definition 2.6.1. A *Sturm sequence* for the polynomial f is a sequence of non-zero polynomials $S_k, \dots S_0$ with $S_k = f, S_{k-1} = f'$ and

$$\alpha_i S_{i+1} = Q_i S_i + \beta_i S_{i-1}$$

for some $Q_i \in D[x], \alpha_i, \beta_i \in D$ and $\deg S_{i-1} < \deg S_i$ where the *Sturm property* $\alpha_i \beta_i < 0$ is satisfied for all $i = 1, \dots, k-1$

The famous theorem from Sturm counts the real roots of a polynomial in some interval. A proof can be found in [Yap00, §7.3]. For a sequence of real numbers $I := (a_k, \dots, a_0)$, let Var I again denote the number of sign variations of I, i.e. the number of sign switches from plus to minus or vice versa, ignoring zeroes.

Theorem 2.6.2 (Sturm). Let $f \in \mathbb{R}[x]$ have Sturm sequence $S_k, \dots, S_0$, and $a, b \in \mathbb{R}, a < b$ such that $f(a) \neq 0 \neq f(b)$. The number of real roots in $[a, b]$ (counted without multiplicities) is given by

$$\mathrm{Var}(S_k(a), \dots, S_0(a)) - \mathrm{Var}(S_k(b), \dots, S_0(b)).$$

For our applications, we are especially interested in the total number of real roots. This can be computed as follows:

Corollary 2.6.3. Let $f \in \mathbb{R}[x]$ have the Sturm sequence $S_k, \dots, S_0$, and let $s_k, \dots, s_0$ be the sequence of leading coefficients. The total number of real roots of f is

$$\sum_{i=0}^{k-1} \epsilon_i, \quad \text{with} \quad \epsilon_i = \begin{cases} 0 & \text{if } \deg(S_{i+1}) - \deg(S_i) \text{ is even} \\ \mathrm{sgn}(s_{i+1} s_i) & \text{if } \deg(S_{i+1}) - \deg(S_i) \text{ is odd} \end{cases}$$

Proof. We choose a large interval $[a, b]$ such that all real roots of any S_i are contained in (a, b). $\mathrm{Var}(S_k(a), \dots, S_0(a))$ and $\mathrm{Var}(S_k(b), \dots, S_0(b))$ are then determined by the leading coefficients – more precisely,

$$\mathrm{Var}(S_k(b), \dots, S_0(b)) = \mathrm{Var}(s_k, \dots, s_0)$$

and

$$\mathrm{Var}(S_k(a), \dots, S_0(a)) = \mathrm{Var}(\delta_k s_k, \dots, \delta_0 s_0)$$

with $\delta_i = (-1)^{\deg(S_i)}$.

We run through both sequences in parallel and we count 1 if there is a sign variation from $S_{i+1}(b)$ to $S_i(b)$, but not from $S_{i+1}(a)$ to $S_i(a)$, we count -1, if we find a sign variation from $S_{i+1}(a)$ to $S_i(a)$, but not from $S_{i+1}(b)$ to $S_i(b)$, and we count 0 otherwise. A simple case distinction yields the formula. □

As an easy corollary from Theorem 2.5.5, one can show that the sequence of regular subresultants of f and f' can be transformed into a Sturm sequence by only multiplying some entries with -1. But it is not that easy to give an explicit

formula – Hong [Ho96] presents a solution for this conversion. Yap [Yap00, §7.1.1] describes an algorithm to transform an arbitrary polynomial remainder sequences into a Sturm sequence. These method have the drawback that they have no good specialisation properties. Calculations are necessary for each x-value α to obtain the Sturm sequence.

Our approach follows Gonzalez-Vega et al. [GL+98]. Their definition of *Sturm-Habicht sequences* does not completely satisfy the Sturm property, but we are still in position to count the real roots of polynomials. Furthermore, the sequence can be computed for a polynomial with parameters, and the specialisation of the parameters always yields a valid Sturm-Habicht sequence again:

Definition 2.6.4. Let $\delta_k := (-1)^{k(k+1)/2}$. For f as above and $k \in \{0, \ldots, n\}$, the *kth 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 *kth principal Sturm-Habicht coefficient* is defined as

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

for $k = 0, \ldots, n$.
The *kth coprincipal Sturm-Habicht coefficient* is defined as

$$\mathrm{costha}_k(f) := \mathrm{coef}_{k-1}(\mathrm{StHa}_k(f))$$

for $k = 1, \ldots, n$.

We call $(\mathrm{StHa}_n(f), \ldots, \mathrm{StHa}_0(f))$ the *Sturm-Habicht sequence of* f. The principal and coprincipal Sturm-Habicht sequences are defined the same way.
Observe that many results from the previous section still hold for Sturm-Habicht polynomials instead of subresultant. We summarise them for completeness.

Corollary 2.6.5. Let k be the degree of $\gcd(f, f')$.

1. $k = \min\{k \in \{0, \ldots, n-1\} \mid \mathrm{stha}_k(f) \neq 0\}$
2. $\mathrm{StHa}_k(f) = \gcd(f, f')$
3. If β is the only root of f over $\overline{Q(D)}$, then

$$\beta = -\frac{\mathrm{costha}_k(f)}{k \cdot \mathrm{stha}_k(f)}.$$

4. Let $\varphi : D \to D'$ be a homomorphism of domains with $\mathrm{lc}(f) \notin \ker \varphi$. Then, for any $i \in \{0, \ldots, m-1\}$

$$\varphi(\mathrm{StHa}_i(f)) = \mathrm{StHa}_i(\varphi(f)).$$

If all Sturm-Habicht polynomials are regular (defined in the same way as for subresultants), it follows from (2.5.5) that the Sturm-Habicht sequence has the Sturm property. In general, this is not true:

Example. Consider $f = 2x^4 + 4x^3 + 3x^2 + x$. One computes the principal Sturm-Habicht coefficients $\mathrm{stha}_4, \ldots, \mathrm{stha}_0$ which are $1, 8, 0, 0, -8$. This means that the sequence of regular Sturm-Habicht polynomials is

$$(\mathrm{StHa}_4, \mathrm{StHa}_3, \mathrm{StHa}_0) = (f, f', \mathrm{res}(f, f')).$$

We have that

$$64f = (16x + 8)f' + (-8)$$

or, equivalently

$$64\mathrm{StHa}_4 = (16x + 8)\mathrm{StHa}_3 + \mathrm{StHa}_0.$$

So, $\alpha = 64, \beta = 1$ and according to Definition 2.6.1, this is not a Sturm sequence.

Though Theorem 2.6.2 is not directly applicable for Sturm-Habicht sequences, we can still count the total number of roots, similar to Corollary 2.6.3. Therefore, we define the following function:

Definition 2.6.6. For a sequence $I := (a_0, \ldots, a_n)$ of real numbers with $a_0 \neq 0$, define

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

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 ith subsequence of I, define

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

Now applying C on the principal Sturm-Habicht sequence of f gives the result:

Theorem 2.6.7. For $f \in \mathbb{R}[x]$ with $\deg f = n$, we have:

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

This is taken from [GN02], the proof needs some deeper results from subresultant theory, it can be found in [GL+98]. Note that the result follows from Corollary 2.6.3 if all Sturm-Habicht polynomials are regular. Also, the total number of real roots only depends on sign of the principal Sturm-Habicht coefficients.

We explain how Sturm-Habicht sequences help to give information about geometric properties of algebraic curve. Let $f \in \mathbb{Z}[x, y]$ be such an curve. We assume that it is primitive, and that the principal Sturm-Habicht sequence $(s_n, \ldots, s_0) := (\mathrm{stha}_n(f), \ldots, \mathrm{stha}_0(f))$ is known. Each element of the sequence is a polynomial in x. Now, we fix some x-coordinate α. To learn about the number of points supported by α, we need to evaluate the signs of $(s_n(\alpha), \ldots, s_0(\alpha))$ which are needed in the count-function C. For information about critical points, we are interested in the degree of f_α and f'_α, so need to evaluate the signs of $s_0(\alpha), s_1(\alpha), \ldots$ until we encounter the first non-zero element, according to Corollary 2.6.5. But these signs are already known from the first step, so we have the nice property that computing the real points over α gives the degree of the gcd of f_α with its derivative for free, and only the principal Sturm-Habicht coefficients suffice to compute that quantities. We will show in Section 4.2 how these two numbers help the Bitstream Descartes method to isolate the real roots of f_α even if the polynomial contains one multiple root.

Chapter 3

Description of the Algorithm

We start with a repetition of the problem statement: For a real algebraic plane curve C, induced by a square free integer polynomial f, we want to answer the following queries for arbitrary $\alpha \in \mathbb{R}$:

- Does the curve contain the vertical line $x = \alpha$ as a component? If it does, the following questions refer to the primitive part of the curve.
- How many points on the curve C have x-coordinate α? We will call this number n_α.
- How many arcs of the curve converge to the vertical asymptote $x = \alpha$ in direction $+\infty$ and $-\infty$, from the left and from the right?
- For $i \in \{1, \ldots, n_\alpha\}$: Is the point (α, β) an event point, where β is the ith point of C over α (in increasing order)?
- For $i \in \{1, \ldots, n_\alpha\}$: How many arcs are incident to (α, β) from the left, and from the right, where β is defined as above?
- For $i \in \{1, \ldots, n_\alpha\}$ and $\epsilon > 0$: Find an interval containing β of size smaller than ϵ, where β is defined as above.

Our algorithm analyses the curve such that the queries above can be answered efficiently. It proceeds in two steps:

- **Projection phase:** Compute a finite set containing all event x-values of the curve.
- **Extension phase:** For each element α of the computed set, create a data structure collecting geometric information of the curve at x-value α.

The data structure to create will be defined formally in Section 3.1. Basically, it contains the answers for all queries concerning an event value α, plus isolating intervals which can be further refined for approximation.

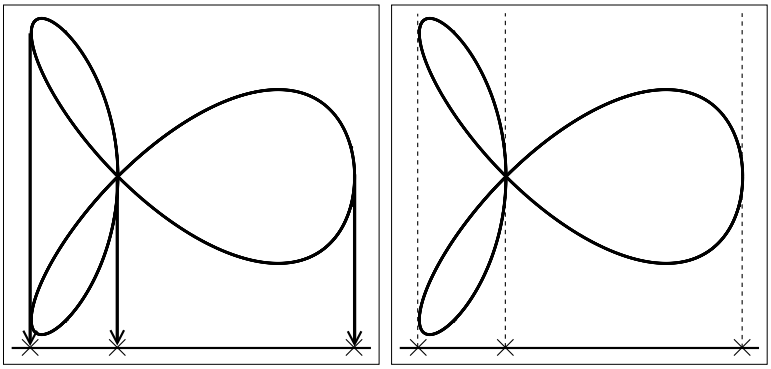

Figure 3.0.1: Illustration of projection and extension phase

The projection phase of the algorithm is treated in Section 3.2. We follow the usual approach of computing the resultant of the polynomial symbolically and isolating its real roots. We give an overview of our new approach for the extension phase in Section 3.3, postponing the details to the subsequent two chapters. We describe next how queries can be answered efficiently for non-critical values (Section 3.4). For the approximation query, we introduce a generalised Descartes algorithm in Section 3.5 and we describe how to apply this algorithm when approximating at event x-values (Section 3.6). For the whole analysis, we only consider the primitive part of f and explain how to analyse curves with vertical line components in Section 3.7.

3.1 The data structure

Our data structure is a sequence of so called *vert-line objects*. These objects store information about a curve at a certain value α and they are arranged in increasing order of α:

Definition 3.1.1. Let $f \in \mathbb{Z}[x, y]$. A *vert-line object* for α is a seven-tuple

$$(\alpha, \texttt{vert\_comp}, \texttt{local\_degree}, \texttt{number\_of\_points}, \texttt{number\_of\_arcs\_lr}, \texttt{asym\_numbers}, \texttt{points})$$

such that

- $\alpha \mathrel{\widehat{=}} (R, I)$ is a simple interval representation of a real algebraic number.
- `vert_comp` is a flag denoting whether or not the vertical line $x = \alpha$ is a component of f.
- `local_degree` is the degree of the polynomial f_α.
- `number_of_points` is the number of points on f supported by α.

- `number_of_arcs_lr`=(l, r) is a pair of integers, l denoting the number of points supported by $\alpha - \epsilon$, r denoting the number of points supported by $\alpha + \epsilon$, where ϵ is arbitrarily small. Equivalently, l is the number of arcs that are either incident to any point at α from the left, or converging to the vertical asymptote $x = \alpha$ from the left. The same holds for r from the right.
- `asym_numbers`=(`plus_left`,`plus_right`,`minus_left`,`minus_right`) is a tuple of integers. `plus_left` denotes the number of arcs that converge to $(\alpha, +\infty)$ from the left side. The other three values have the corresponding meaning.
- `points` is a sequence of length `number_of_points`. Its elements are triples

  (`approx`, `incidence_numbers`, `event`),

  containing further information about the points supported by α, in increasing order of y.
  - `approx` is an interval such that $(f, \alpha,$ `approx`$)$ is an algebraic interval representation of that point.
  - `incidence_numbers`=(l, r) is a pair of integers, denoting how many arcs are incident to this point from the left, or from the right respectively.
  - `event` is a flag indicating whether or not this point is an event point.

Obviously, if such a vert-line object exists for α, all queries listed above can be answered in constant time, except for further approximation.

Example. Consider the curve

$$f = (xy - 1)(x - y)(x + y)(x^2 + y^2 - 2)$$

illustrated in Figure 3.1.1. We want to create the vert-line for $\alpha = 0$. It should be clear that the elements of the seven-tuple take the following values,

α	=	0
`vert_comp`	=	`false`
`local_degree`	=	4
`number_of_points`	=	3
`number_of_arcs_lr`	=	(5,5)
`asym_numbers`	=	(0,1,1,0)
`points`	=	((I_1,(1,1),false), (I_2,(2,2),true), (I_3,(1,1),false))

where I_1, I_2, I_3 are isolating intervals for $-\sqrt{2}, 0, \sqrt{2}$ respectively.

3.2 Identification of event values (projection phase)

As we want to create vert-line objects for each event value, the question arises how to find them algorithmically. Fortunately, the resultant is a very suitable tool for that. Recall that the resultant of f and g is defined to be the zeroth subresultant. From now, we use the following notation.

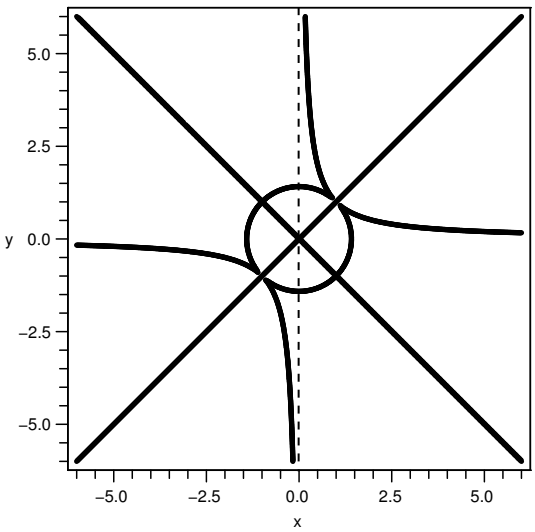

Figure 3.1.1: The graph of $f = (xy-1)(x-y)(x+y)(x^2+y^2-2)$.

Definition 3.2.1. For polynomials f and g in $K[x_1, \ldots, x_n]$, we write $\operatorname{res}(f, g, x_i)$ to denote the resultant of f and g, considered as univariate polynomials in x_i. In particular, $\operatorname{res}(f, g, x_i) \in K[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$. We define $\operatorname{Sres}_i(f, g, y), \operatorname{sres}_i(f, g, y)$ for all possible i in the same way.

The next well-known lemma (for instance in [CLO92, §3.5, Prop.8]) is a special case of our results from Section 2.5:

Theorem 3.2.2. Two polynomials $f, g \in \mathbb{R}[x]$ have a common factor with positive degree if and only if $\operatorname{res}(f, g) = 0$

Proof. The first condition is equivalent to $\deg_y(\gcd(f, g)) \geq 1$. But Corollary 2.5.2 tells us that

$$\deg_y(\gcd(f, g)) = \min\{k \mid \operatorname{sres}_k(f, g, y) \neq 0\}$$

and this yields the equivalence at once. □

Corollary 3.2.3. For polynomials $f, g \in \mathbb{R}[x, y]$: If (α, β) is an intersection point of f and g, then $\operatorname{res}(f, g, y)(\alpha) = 0$.

Proof. If (α, β) is intersection point, we have that $f_\alpha(\beta) = g_\alpha(\beta) = 0$, and therefore f_α and g_α have the common factor $(x-\beta)$, so $\operatorname{res}(f_\alpha, g_\alpha) = 0$. By the specialisation property (Theorem 2.5.6), it is $\operatorname{res}(f_\alpha, g_\alpha) = \operatorname{res}(f, g, y)(\alpha)$. □

For $g := D_y f$, this tells us that the resultant vanishes for x-coordinates of critical points. If $x = \alpha$ is vertical asymptote of f, the leading coefficient of f vanishes. It follows that $\operatorname{res}(f, D_y f, y)(\alpha) = 0$ also in this case, since the first column of the Sylvester matrix only contains zeros. We conclude

Theorem 3.2.4. If α is a critical x-value of f, then $\mathrm{res}(f, D_y f, y)(\alpha) = 0$.

This is only a necessary condition: It can happen that the resultant has roots which do not support critical points, if f and $D_y f$ meet in complex points with real x-coordinate, or if the leading coefficient of the polynomial vanishes without causing a (real) asymptotic arc at this x-coordinate.

Furthermore, $R := \mathrm{res}(f, D_y f, y) \neq 0$ for square free f because f and $D_y f$ have no common component. This shows that R only has finitely many roots, and the set of these roots contains all event x-values. The idea is now to find the roots of R and build a vert-line object for each root. Since the roots of R are algebraic numbers, we represent them in interval representation. Algorithm 2.4.7 describes how to find the real roots of R. For a repetition, we first factorise R into square free polynomials $R_1, \ldots, R_r$ where R_i contains precisely the roots of R with multiplicity i. Then, we isolate the real roots of each R_i, using the Descartes method. All roots are then merged into one increasing sequence.

We could define an algorithm for the projection phase in a straightforward fashion by computing the resultant and isolating its roots. Since we will make use of the principal and coprincipal Sturm-Habicht coefficients in the extension phase, and since the zeroth principal Sturm-Habicht coefficient is an associate to the resultant R, we save time by computing it already in the projection phase:

Algorithm 3.2.5 (Projection phase).
Input: Square free and primitive polynomial $f \in \mathbb{Z}[x, y]$.
Output: $\alpha_1, \ldots, \alpha_n$ roots of the resultant $\mathrm{res}(f, D_y f, y)$, $\mathrm{stha}_0(f), \ldots, \mathrm{stha}_m(f)$ principal Sturm-Habicht coefficients, $\mathrm{costha}_1(f), \ldots, \mathrm{costha}_m(f)$ coprincipal Sturm-Habicht coefficients.

1. Compute the principal and coprincipal Sturm-Habicht coefficients. In particular, this gives $R := \mathrm{res}(f, D_y, y) \sim \mathrm{stha}_0(f)$.
2. Factorise R into $R_1, \ldots, R_r$ using the square free factorisation method and isolate the real roots with the Descartes method (Algorithm 2.4.7).
3. Merge the roots of all R_i's into an increasing sequence.

The computation of the Sturm-Habicht coefficients can either be done by evaluating the appropriate minors of Sylvester submatrices, or by using pseudo-division-based approaches that exploit the relation of Sturm-Habicht polynomials with the Euclidean algorithm. We postpone a more detailed discussion of both variants to Chapter 6.

3.3 Building vert-line objects (extension phase)

The projection phase returns a set of x-values $\alpha_1, \ldots, \alpha_n$ where all event x-values are included. The next step is the extension phase, where we construct a vert-line object for each α_i.

We describe a new algorithm for the extension phase. It is structured in two steps: First, we apply an algorithm called `Generic Extension`: It tries to create

the vert-line objects by working directly in the original coordinate system. For that, it relies on some genericity assumptions for the curve that simplify the analysis, for example the curve must not have vertical asymptotes, and also covertical critical points are forbidden. But the algorithm detects unfavourable situations during execution and reports a failure in this cases. For real root isolation of f_{α_i}, it uses a new variant of the Bitstream Descartes method which can handle one multiple root. This is the main step to get all necessary data for the vert-line objects. All details are described in Chapter 4.

If `Generic Extension` reports a failure, a second approach must be applied: The curve is then analysed in a different coordinate system to get a, roughly speaking, more generic position (where features as vertical asymptotes or covertical critical points do not occur). Algorithmically, this change of coordinates correspond to a *shear* of the curve, i.e. we transform f into a new polynomial $\mathfrak{S}_s f = f(x + sy, y)$ with some *shear factor* s. With the newly generated $\mathfrak{S}_s f$, we apply an algorithm called `Nongeneric Extension`: It tries to create the vert-line objects of the original curve by using the sheared curve. The algorithm first analyses the curve $\mathfrak{S}_s f$ in a similar fashion as `Generic Extension` and a set of *sheared-line objects* (which are slightly modified vert-line objects) is created. This substep can also fail, and the algorithm is able to detect these situations. Using the α_i's and the sheared-line objects, `Nongeneric Extension` applies a new technique to create the vert-lines of the original curve. Details of all substeps are described in Chapter 5.

Algorithm 3.3.1 (Extension phase).
Input: $f \in \mathbb{Z}[x, y]$ square free and primitive, $\alpha_1, \ldots, \alpha_n$
Output: Vert-line object for each α_i.

1. Try to apply `Generic Extension`. In case of success, return the vert-line objects.
2. In case of a failure, repeat choosing a shear factor s and apply `Nongeneric Extension` until it runs successful. Return the vert-line objects.

We are aware that the algorithm is not conform to the idea of randomised algorithm where unfavourable inputs are prevented through initial randomisation. Though we decided to use `Generic Extension`, because it speeds up the analysis for a large class of polynomials. For instance, it only needs to compute a single resultant whereas `Nongeneric Extension` needs two additional resultants to work with. Of course, cancelling the call of `Generic Extension` in the extension phase would give a second variant which is more independent of the initial coordinate system.

3.4 Queries for non-critical values

We have explained how to detect event values of primitive curves and how to create vert-line objects for them. We assume for the remainder of the chapter that the vert-lines are created and we turn to the question how the queries (listed at

the beginning of the chapter) for the curve can be answered using these vert-line objects. We start with the case a query must be answered for an x-coordinate where no vert-line exists. We assume that all x-coordinates are represented in simple interval representation.

Let γ be such an x-coordinate. γ is non-critical since critical x-values are roots of the resultant of f and $D_y f$, and a vert-line object is created for them. Most queries are thus easy to answer: There is certainly no event point supported by γ and no vertical asymptote. Every point has one incident arc from the left and one from the right. Only two queries remain: The number of supported points, and an ϵ-approximation of the ith point.

The number of points on the curve with $x = \gamma$ can be answered using the vert-line objects: One simply finds the smallest event x-value that is greater than γ, if existing. Let α be the x-coordinate of this vert-line. The number of points supported by $\alpha - \epsilon$ is stored in the `number_of_arcs_lr` field of the vert-line, and this number is equal to the number of points supported by γ (Theorem 2.2.29). If such an event x-value does not exist, γ is greater than any event x-value, and the vert-line for the greatest event x-value α contains the number of points supported by $\alpha + \epsilon$, which is equal to the number of points supported by γ.

For the approximation, first note that f_γ is square free, because multiple roots would cause a critical point of f. Here, we exploit that we have created vert-line objects for all critical x-values of the curve, and not only for event x-values. We can isolate the real roots of f_γ with the Bitstream Descartes method to obtain isolating intervals of its roots. Let I be the ith isolating interval. The y-coordinate can be represented in algebraic interval representation (f, γ, I) and we know from Section 2.4 how to refine this number to arbitrary precision.

3.5 A generalised Descartes algorithm

It remains the question how queries can be answered for x-coordinates that own a vert-line object. For the approximation of those values, we need a method of the (Bitstream) Descartes method that can handle multiple roots. As the same principle is also applied in the extension phase in the subsequent chapters, we develop the general idea next.

Recall Algorithm 2.3.12, the Descartes algorithm for square free polynomials. For a more compact description, we assume from now that none of the chosen split points is a root of the input polynomial. As we will always work with approximations of the coefficients, this is no restriction, since the Bitstream Descartes method avoids to choose roots as split points through randomisation. But our idea also applies for the usual Descartes method if this special case is handled properly.

Algorithm 2.3.12 is implemented with a depth-first-search strategy, we change the traversal strategy to breadth-first-search in the generalised method. As main novelty, we relax the termination condition that the leaves of the Descartes tree must be marked with either one or zero. Instead the termination condition must be

specified by the application. The markings of the leaves are returned as additional data. When the termination condition is satisfied, an additional post-processing step allows to manipulate the set of leaves before returning it.

Algorithm 3.5.1 (Generic Descartes).
Input: Polynomial f, initial interval J
Output: A sequence of pairs (I, n) such that I is an isolating interval for a root of f with v sign variations.

1. Compute the sign variation v in J. If $v = 0$, return the empty sequence. Otherwise initialise the sequence `leaves` as $[(J, v)]$.
2. While the termination condition is not satisfied, repeat:
   i. If `leaves` is empty, return the empty sequence
   ii. Delete the first element $((c, d), v)$ from `leaves`.
   iii. If $v = 1$, append $((c, d), n)$ to the end of `leaves` and continue.
   iv. Otherwise, chose a split point m, obtaining the intervals $I_1 = (c, m)$ and $I_2 = (m, d)$. Compute v_1, v_2, the sign variations of I_1, I_2.
   v. If $v_1 > 0$, append (I_1, v_1) to `leaves`. If $v_2 > 0$, append (I_2, v_2) to `leaves`.
3. Post-process `leaves`.
4. Shift `leaves` cyclically such that the intervals are ordered in increasing order.
5. Return `leaves`.

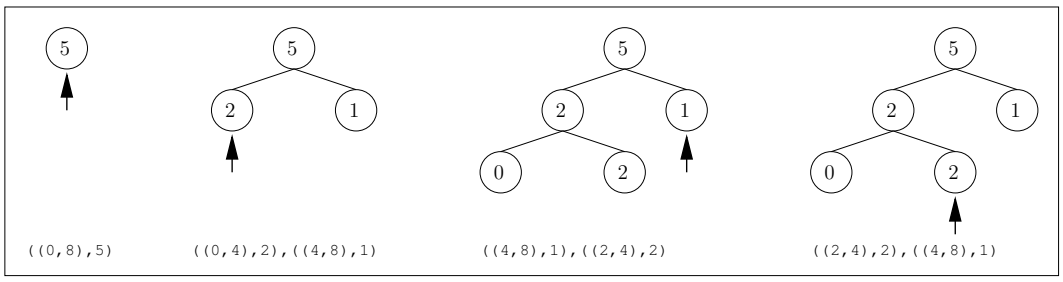

Figure 3.5.1: Example for `Generic Descartes`. It is assumed that the initial interval is $(0, 8)$ and the split point is always chosen in the middle. The arrow marks the head in the `leaves` sequence, the next processed interval.

At each stage, the `leaves` sequence corresponds to the leaves in the Descartes tree with non-zero sign variation, and the intervals are sorted in increasing order up to some cyclic shift (Figure 3.5.1). The BFS version of Algorithm 2.3.12 is now easily obtained: the termination condition is that all intervals in `leaves` have exactly one sign variation. The post-process stage (Step 3 in the algorithm) can remain empty in this case.

We also define a `Generic Bitstream Descartes` algorithm which is the analogue to the `Generic Descartes` version for approximate calculations. Recall that

for the Bitstream Descartes, the number of sign variations might not be determinate, so there is a set of possible sign variations instead.

Algorithm 3.5.2 (Generic Bitstream Descartes).
Input: Polynomial f, initial interval J
Output: A sequence of pairs (I, V) such that I is an isolating interval for a root of f with a set V of possible sign variations.

1. Compute the possible sign variations V in J. If $V = \{0\}$, return the empty sequence. Otherwise initialise the sequence `leaves` as $[(J, V)]$.
2. While the termination condition is not satisfied, repeat:
   i. If `leaves` is empty, return the empty sequence
   ii. Delete the first element $((c, d), V)$ from `leaves`.
   iii. If $V = \{1\}$, append $((c, d), V)$ to the end of `leaves` and continue.
   iv. Otherwise, chose a split point m, obtaining the intervals $I_1 = (c, m)$ and $I_2 = (m, d)$. Compute V_1, V_2, the sign variations of I_1, I_2.
   v. If $V_1 \neq \{0\}$, append (I_1, V_1) to `leaves`. If $V_2 \neq \{0\}$, append (I_2, V_2) to `leaves`.
3. Post-process `leaves`.
4. Shift `leaves` cyclically such that the intervals are ordered in increasing order.
5. Return `leaves`.

In our algorithm, we only use instances of the `Generic Bitstream Descartes` algorithm. However, we decided to describe their exact version first as instances of `Generic Descartes` and explain how to apply them in a Bitstream version as a second step (as in the next section). We think that this helps for a better understanding of the presented algorithm.

3.6 Approximation of event points

Suppose that there exist a vert-line object for α (this is always the case for critical values). We can easily answer most queries since the vert-line provides the suitable information. Only the approximation of the ith point supported by α is not immediate.

The vert-line object provides an isolating interval I for the y-coordinate β of the ith point. If that point is a non-event point, β is a simple root and I can be refined by bisection. The remaining question is how to refine the interval for event points. If $I = [c, d]$ and the sign at the boundaries of c and d differs, we can again use bisection to refine the interval for that event point (this applies for each non-event point of the curve, since vertical flexes have this property). The difficult part arises if the signs at the boundaries are equal.

For shorter notation, set $g := f_\alpha$, and let β be the y-coordinate of the event point with I as isolating interval. Since the signs at the boundaries of the interval do not differ, β must have even multiplicity (this follows from the proof of Theorem 2.3.11).

If the Descartes algorithm is started for the polynomial g and isolating interval I, the interval is split into I_1 and I_2, and two outcomes are possible: If one of the intervals has zero sign variations, the other one accommodates the root and we did successfully refine the interval. Otherwise, both intervals have at least one sign variation (in fact, they must also have at least sign variations), but is is sure that one of them does not contain a real root because I is isolating. We call splits of this kind *noisy*. W.l.o.g. let I_1 contain the root. Since the algorithm cannot decide which interval contains the root, it must refine both, and after finitely many steps, all intervals which are subsets of I_2 count zero as sign variation, in other words, no subset of I_2 appears in the `leaves` sequence of the algorithm (we say that the noisy split is *hushed*).

During the execution of the algorithm, also the interval containing α was refined several times, and it might be the case that more noisy splits were produced. But as before, these splits will hush after finitely many steps, and the total number of noisy splits is finite, since the sign variation drops in each noisy split (Theorem 2.3.13). It follows that the `leaves` sequence eventually consists of one single interval and this is taken as the refined version of the isolating interval (Figure 3.6.1.

In summary, to refine a root (f, α, I) in algebraic interval representation, we call `Generic Descartes` for f_α and I. The termination condition is that the `leaves` sequence contains exactly one interval that is a proper subset of I (to avoid that the algorithm terminates without doing anything). The post-process step can remain empty.

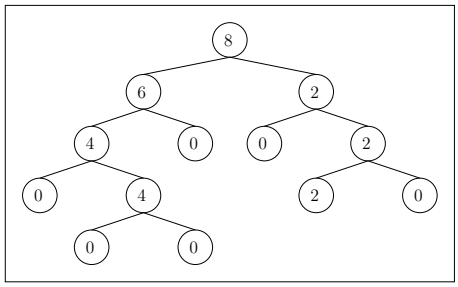

Figure 3.6.1: An example of a noisy split. Note that further refinements are easy since the number of sign variations has fallen to 2, and there cannot be more noisy splits.

This algorithm can be directly transferred into the bitstream model. Each noisy split must eventually hush, since the interval that contains no real root will eventually count zero sign variations also in the approximated version [EK+05]. So we can apply `Generic Bitstream Descartes` (Algorithm 3.5.2) with the same termination condition as above.

At this point, we make an important remark: Let $f = \sum_{i=0}^{n} r_i(x)y^i$ be the

defining polynomial of the curve, and assume that f_α is passed to any instance of `Generic Bitstream Descartes`. The Bitstream Descartes algorithm needs to compute a lower bound of the leading coefficient of its input polynomial. It is not always the case that $r_n(\alpha)$ is the leading coefficient of f_α because $r_n(\alpha) = 0$ is possible. Since the Bitstream Descartes only uses approximations of $r_i(\alpha)$, it cannot decide whether this coefficient is zero or not. This means, whenever the Bitstream Descartes method is applied for f_α, the degree of that polynomial must be known beforehand. That is the reason why the `local_degree` field is included in the vert-line object.

3.7 Curves with vertical components

We have explained how to answer all queries listed at the beginning for primitive input curves. We describe next how this can be extended to non-primitive curves.

In a very first step of the algorithm, we divide out the content of the input polynomial and analyse the primitive part. Lemma 2.2.6 tells us that the roots of the content are exactly the vertical lines of the curves. Since each root α of the content is an event x-value, there must exist a vert-line object for α where the `vert_comp` flag is set. If the vert-line object for α already exists from the analysis of the primitive part, we can directly set the flag, otherwise we must first create the corresponding vert-line. This is easy with the results from Section 3.4.

The complete algorithm looks as follows.

Algorithm 3.7.1.
Input: Square free polynomial $f \in \mathbb{Z}[x, y]$
Output: Vert-line objects (at least) for all critical x-values of f

1. Decompose $f = \mathrm{cont}(f)\mathrm{pp}(f)$.
2. **Projection phase:** Call Algorithm 3.2.5 on $\mathrm{pp}(f)$.
3. **Extension phase:** Call Algorithm 3.3.1 on $\mathrm{pp}(f)$.
4. Isolate the roots of $\mathrm{cont}(f)$ and create vert-line objects at these positions, if necessary. Set the `vert_comp` flag at these x-coordinates.

The idea is very simple: We take out the vertical line components, analyse the primitive curve, and add the vertical lines at the end. We did not give details for the extension phase so far. We continue with this subject in the next two chapters.

Chapter 4

Extension Phase for "Sufficiently Generic" Curves

4.1 Overview

The goal of this chapter is the definition of the algorithm `Generic Extension` which we have already sketched in Section 3.3: For a square free and primitive polynomial f and a set of values $\alpha_1, \ldots, \alpha_n$ (where all event x-values of f are contained), it computes vert-line objects for the α_i's.

The algorithm tries to analyse the curve directly in the original coordinate system. It combines exact information about the polynomial f_α, obtained from the Sturm-Habicht sequence with the Bitstream Descartes method to produce the vert-line objects in an exact and efficient way. There is no guarantee that the algorithm succeeds, but it detects awkward situations during the execution and reports a failure in these cases. In order to work, it requires certain genericity condition on the curve, but we will not define these properties formally. Instead, we give two guarantees which are related with the following two well-defined genericity conditions:

Definition 4.1.1. Let $\mathbb{K}$ be the field $\mathbb{R}$ or $\mathbb{C}$. A polynomial $f \in \mathbb{R}[x, y]$ is in $\mathbb{K}$-generic position if

- The leading coefficient of f, considered as polynomial y is a constant (we call this *y-regularity*) and
- For all $\alpha \in \mathbb{R}$, the polynomial $f_\alpha \in \mathbb{R}[y]$ has at most one multiple root in $\mathbb{K}$.

Note that $\mathbb{R}$-genericity implies that f has no vertical asymptote (from Theorem 2.2.26) and no two covertical event points. $\mathbb{C}$-genericity is a stronger condition implying very useful algebraic properties (for instance, this implies that each root of the resultant supports a (real) critical point and the y-coordinate of this point can be expressed as rational expression according to (2.5.6)). The algorithms

from Gonzalez-Vega and Necula [GN02], and Seidel and Wolpert [SW05] check $\mathbb{C}$-genericity of the curves.

Proposition 4.1.2. `Generic Extension` satisfies the following properties:

- If the input curve is not $\mathbb{R}$-generic, `Generic Extension` reports a failure.
- If the input curve is $\mathbb{C}$-generic, `Generic Extension` succeeds.

$\mathbb{R}$-genericity is a must for the algorithm: If two critical points are covertical at some x-coordinate α, there are two multiple roots at f_α. The advancement of the Bitstream Descartes method that we will define cannot handle these cases. The presence of vertical asymptotes complicates the computation of the incidence numbers and is therefore undesired as well. The second property of Proposition 4.1.2 shows that the algorithm indeed succeeds for a large class of polynomials. Curves which are $\mathbb{R}$-generic but not $\mathbb{C}$-generic lie in a grey zone. The algorithm might succeed or not, and in fact, this can even change from instance to instance, since random choices are made during the algorithm.

As a first step, `Generic Extension` verifies that the input curve is indeed y-regular,or reports a failure otherwise. Recall from Definition 3.1.1 that the vert-line objects are seven-tuples with entries

(α, `vert_comp`, `local_degree`,`number_of_points`, `number_of_arcs_lr`, `asym_numbers`, `points`)

For each α_i, a vert-line object is created. The `asym_numbers` entry is set to $(0,0,0,0)$ and `local_degree` is set to $\deg_y f$ for each vert-line, since the curve is y-regular. The `vert_comp` flag is set to false because the curve is primitive.

The other entries in the vert-line objects are not that obvious to compute. The remaining sections of this chapter explain in detail how `Generic Extension` computes this data.

4.2 Counting real roots

This section discusses how to compute the data field `number_of_arcs_lr` and describes two variants for the calculation of the `number_of_points` field.

For the data field `number_of_arcs_lr`, the number of points supported by $\alpha - \epsilon$ and $\alpha + \epsilon$ must be computed, where ϵ is arbitrarily small. We only describe the former case, the latter is analogous. As the number of supported points does not change between critical points (Lemma 2.2.29), it is enough to count the real roots over any x-coordinate between α_{i-1} and α_i. `Generic Extension` chooses a rational number r in that interval and applies Descartes method for the (square free) polynomial f_r. The number of isolating intervals gives the solution.

For the `number_of_points`, we will fist handle the case that α_i is a simple root of $R = \mathrm{res}(f, D_y f, y)$. This can be decided easily because we have factorised R into

square free factors $R_1, \ldots, R_r$, and the simple roots of R are exactly the roots of R_1.

The geometric situation at simple roots is particulary simple, as the following theorem demonstrates.

Theorem 4.2.1. Let $f \in \mathbb{R}[x, y]$ and α root of $R := \mathrm{res}(f, D_y f, y)$. If α is a simple root of R, then there exists only one critical point p with x-value x_0 and p is a non-singular x-extreme point.

For the proof, we use the following lemma from Wolpert. A proof can be found in [Wol02, §4.1.1].

Lemma 4.2.2. Let (α, β) be an intersection point of two polynomials f and g. Then α is a multiple root of the resultant of f and g if and only if

$$(D_x f \cdot D_y g - D_y f \cdot D_x g)(\alpha, \beta) \cdot \mathrm{sres}_1(f, g, y)(\alpha)$$

vanishes.

Proof of Theorem 4.2.1. Let α be a simple root of $\mathrm{res}(f, D_y f, y)$. Thus there is some $\beta \in \mathbb{C}$ such that $f(\alpha, \beta) = D_y f(\alpha, \beta) = 0$. From Lemma 4.2.2, it follows that

$$(D_x f \cdot D_{yy} f - D_y f \cdot D_{yx} f)(\alpha, \beta) \cdot \mathrm{sres}_1(f, D_y f, y)(\alpha) \neq 0$$

since α is a simple root. By assumption, $D_y f(\alpha, \beta) = 0$, so the term simplifies to:

$$(D_x f \cdot D_{yy} f)(\alpha, \beta) \cdot \mathrm{sres}_1(f, D_y f, y)(\alpha) \neq 0$$

The first subresultant does not vanish, so the gcd has degree one:

$$\gcd(f_\alpha, D_y f_\alpha) = x - \beta$$

and thus (α, β) has no covertical critical point. Moreover, $D_x f(\alpha, \beta)$ does not vanish, so this point is not singular. Since $D_{yy} f(a, b)$ does not vanish, it must be an x-extreme point. □

Let (l, r) be the `number_of_arcs_lr` entry for α_i. Because there is exactly one x-extreme point at α_i, the two integers differ exactly by two. The number of points over α_i is then $\frac{l+r}{2}$, just the value between l and r since two arcs end in a common point. Hence we have computed `number_of_points`. We point out that the proof of Theorem 4.2.1 also showed that the degree of $gcd(f_{\alpha_i}, f'_{\alpha_i}) = 1$, the algorithm will need this number for the next step (Section 4.3).

If α_i is a multiple root of R, we use the principal Sturm-Habicht coefficients for counting: The number of points on the curve supported by α_i equals the number of real roots of f_{α_i} and Theorem 2.6.7 tells us that

$$\{\beta \in \mathbb{R} \mid f_{\alpha_i}(\beta) = 0\} = C(\mathrm{stha}_n(f_{\alpha_i}), \ldots, \mathrm{stha}_0(f_{\alpha_i})).$$

The specialisation property 2.6.5 says that

$$\mathrm{stha}_i(f_{\alpha_i}) = \mathrm{stha}_i(f)(\alpha_i)$$

and we have computed $\mathrm{stha}_n(f), \ldots, \mathrm{stha}_0(f)$ in the projection phase (see Algorithm 3.2.5). Our extension algorithm `Generic Extension` proceeds as follows for multiple roots α_i: First, it evaluates the signs of the principal Sturm-Habicht coefficients (which are polynomials in x) evaluated at α_i. In particular, this yields the degree of the greatest common divisor of f_{α_i} and f'_{α_i} because this number is equal to the minimal index of a non-vanishing principal Sturm-Habicht coefficient (Theorem 2.6.5). This number is stored temporarily for later usage. Applying Theorem 2.6.7 on the obtained sign sequence returns the number of supported points by α_i.

4.3 Root isolation at critical values

So far, we computed all entries in the vert-line object, except the `points` vector. Recall from Definition 3.1.1 that this is a sequence of triples

(`approx`, `incidence_numbers`, `event`),

where each element holds information about some point supported by α_i: An isolating interval, the number of incident arcs from the left and right, and a flag denoting whether the point is an event point or not. As the number of points over α_i is already known from the last section, the algorithm creates (yet empty) elements for each point. This section introduces a new method how the `approx` field is computed, i.e. how to isolate the real roots of f_{α_i}. As the algorithm works for any univariate polynomial with additional information, we describe it for a general $g \in \mathbb{R}[x]$. Although we want to realise it as an instance of `Generic Bitstream Descartes` (Algorithm 3.5.2) for efficiency, we first describe its exact version to communicate the idea.

For $g \in \mathbb{R}[x]$, the following quantities are assumed to be known:

- m, the number of real roots of g
- k, the degree of the greatest common divisor of g and g'

We want to find isolating intervals of the real roots of g, or report a failure in unfavourable cases.

Assume for the moment that g has at most one multiple real root. If the `Generic Descartes Algorithm` (3.5.1) is applied for g (with some sufficiently big initial interval), it eventually isolates all $m-1$ simple roots in intervals with exactly one sign variation, and there is exactly one further leaf having more than one sign variation. It is clear than that this interval must also contain a root of g and the algorithm terminates and reports the m isolating intervals (see Figure 4.3.1 for an example).

This algorithm does not terminate if g has several multiple real roots, we also need a termination condition in these cases. Therefore, assume for the moment that g has several multiple roots over the complex numbers. It follows that each multiple root of g has multiplicity at most k (a proof follows below). The `Generic Descartes Algorithm` will eventually enclose each multiple root in an isolating interval such that the number of sign variations equals the multiplicity of the root (Theorem 2.3.6). In other words, at some state no leaf in the Descartes tree has a mark greater than k. The algorithm terminates and reports a failure in this situation(see Figure 4.3.2 for an example).

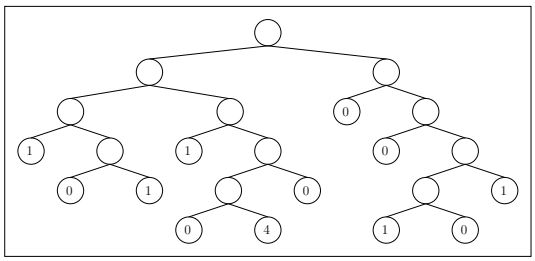

Figure 4.3.1: Assume that we want to find $m = 5$ roots of the polynomial. In this situation, we found 4 simple roots, and there is only one more interval that with greater sign variation. So the algorithm stops at this point, reporting the five non-zero leaves as isolating intervals.

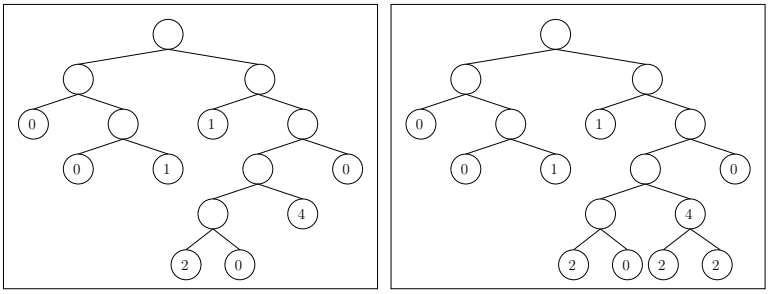

Figure 4.3.2: Suppose that $m = 4$ and $k = 3$. On the left side, there are still two intervals with multiple sign variation. The node marked with four splits into two nodes with mark two (right side), and the maximal sign variation in the leaves also drops down to two. Thus a failure is reported in this situation.

We define the algorithm `m-k-Descartes`, as a specialisation of `Generic Descartes` setting the parameters as follows.

- **Initial interval:** Apply the Fujiwara root bound S from Theorem 2.3.7 for g.

- **Termination condition:** `leaves` has size m with $m-1$ simple roots (condition A), or `leaves` contains only intervals with sign variation at most k (condition B).
- **Post-processing step:** If `leaves` does not satisfy A, report a failure.

Lemma 4.3.1. `m-k-Descartes` terminates for any g.

Proof. We have already argued that condition A is satisfied eventually if g has at most one multiple real root. If g has more than one multiple root over the complex numbers, let $k_1, \ldots, k_s$ denote the multiplicities of the multiple roots $\beta_1, \ldots, \beta_s$ (with all $k_i > 1$). We know that:

$$k = \deg(\gcd(g(y), g'(y))) = \sum_{i=1}^{s} (k_i - 1)$$

Therefore $k_i \leq k$ for all i. During the algorithm, all real β_i are eventually enclosed in intervals that are counted with multiplicity k_i and condition B is satisfied. □

Consider the case that `m-k-Descartes` is applied for polynomial g that has one multiple real root, but several complex roots. The proof shows that both termination conditions A and B are eventually satisfied, and it is not clear whether the algorithm succeeds or not. This depends on which condition is satisfied first.

In all other cases, we can give the following guarantees:

Lemma 4.3.2.

- If g has more than one multiple real root, `m-k-Descartes` reports a failure.
- If g has exactly one multiple root over the complex numbers, `m-k-Descartes` returns the isolating intervals of g.

Proof.

- It is enough to show that condition A is never satisfied in presence of several multiple real roots. But this follows at once, since the algorithm can never find $m-1$ simple roots of g, because there are at most $m-2$.
- We must show that condition B is never satisfied here. As there is at only one complex root β, we know that β must be real. This follows from the fact that β can be expressed as rational expression in the subresultant coefficients of g (2.5.6). Furthermore, it is

  $$\gcd(g, g') = (x - \beta)^k$$

  and therefore, β is root of g with multiplicity $k+1$. The interval containing β has always at least $k+1$ sign variations and assures that condition B is not satisfied.

□

Important remark. If the algorithm succeeds, it does not guarantee that the interval with multiple sign variation contains a multiple root. It is possible that a simple root has imaginary roots in its closer neighbourhood causing a greater number of sign variations in that interval. At least, it is clear that all other intervals contain simple roots.

We formulate the bitstream version next. The previous properties and proofs still apply, since a root with multiplicity v will eventually be enclosed in an isolating interval with the set $\{v\}$ as set of possible sign variations. Thus we can straightly define the algorithm `Bitstream m-k-Descartes` as a specialisation of `Generic Bitstream Descartes` and the following parameters:

- **Initial interval:** Apply the Fujiwara root bound S from Theorem 2.3.7 for g.
- **Termination condition:** `leaves` has size m with $m-1$ intervals that have $\{1\}$ as set of possible sign variation (A), or `leaves` contains only intervals where the maximal possible sign variation is at most k (B).
- **Post-processing step:** If `leaves` does not satisfy (A), report a failure.

We explain how this algorithm is used for the analysis of our curve f: `Generic Extension` applies `Bitstream m-k-Descartes` for each polynomial f_{α_i}. If the real roots are isolated successfully, `Generic Extension` fills the `approx` fields in the `points` sequence of α_i. If a failure is reported, also `Generic Extension` reports a failure. Using Lemma 4.3.2, we can now prove the first part of Proposition 4.1.2:

Lemma 4.3.3. If f is not $\mathbb{R}$-generic, `Generic Extension` reports a failure.

Proof. If f has a vertical asymptote, this was detected in the very first step of `Generic Extension`. Otherwise, there exists some α with two real multiple roots, and `Bitstream m-k-Descartes` reports a failure for f_α. □

We cannot prove the second part of the proposition yet since the algorithm is not completely described. But we can record:

Lemma 4.3.4. If f is $\mathbb{C}$-generic, `Bitstream m-k-Descartes` succeeds for each f_{α_i}.

Proof. If f is $\mathbb{C}$-generic, each f_{α_i} has exactly one multiple root over the complex and the claim follows with Lemma 4.3.2. □

For the remaining part of the algorithm, we can assume that f is $\mathbb{R}$-generic.

4.4 Incident arc counting

From the last step, isolating intervals for any point over α_i are known. and we turn to the question how many arcs are incident from the left and right of any point.

In fact, this assignment is very easy to make with the information computed so far: There are $m-1$ isolating intervals with exactly one sign variation, so the points inside are non-critical. It follows that they have one incident arc from the left and from the right. We also know the total number of arcs entering from the left and right which is stored in the `number_of_arcs_lr` field. Thus the incidence numbers of the remaining point are obtained by subtracting $m-1$ from these two quantities.

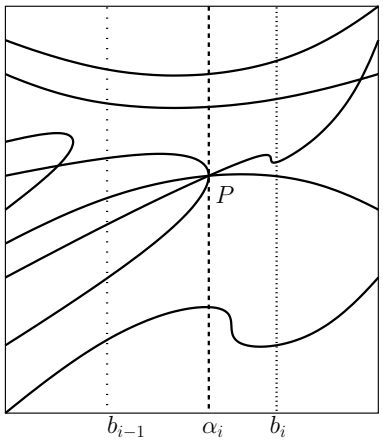

Figure 4.4.1: A singular point with 4 arcs entering and 2 leaving

In Figure 4.4.1, a typical situation is depicted: There are four roots over α_i, and we count seven roots on the left (at some intermediate value b_{i-1}). Thus, four arcs have to be incident to P from the left.

Note that this approach would not work if the curve is allowed to have a vertical asymptote at α or if there is more than one multiple root over α. But these cases are already excluded since the curve is $\mathbb{R}$-generic.

4.5 Finding event points

The last missing data to complete the vert-lines is the `event` flag for any point supported by α_i (we will skip the index and set $\alpha := \alpha_i$ for simplicity). Let I_c denote the isolating interval that has not exactly one sign variation (if there is any). For all other intervals, the flag can immediately be set to `false`, since the point inside is not even critical. We call I_c the *candidate interval* (and the root inside the *candidate point*), denoting that it might or might not contain an event point.

`Generic Extension` first considers the incidence numbers of the candidate point which were computed in the last step. If they are not $(1,1)$, it sets the `event` flag because the point must be an event point (if the incidence numbers are $(0,2)$ or $(2,0)$, the point is x-extreme, and it is singular in all other cases).

If the incidence numbers are $(1,1)$, the candidate point cannot be x-extreme, and three situation are possible for I_c: It may contain a singularity, a vertical flex or a non-critical point (compare Figure 4.5.1). Only in the first case, the candidate point should receive the `event flag`. To distinguish singularities, we explicitly check whether $D_x f$ and $D_y f$ vanish at the candidate point. Doing this directly, i.e. substituting the values α for x and $\beta = (f, \alpha, I_c)$ for y is possible ([ET05] proposes a method for that kind of problem using Sturm-Habicht sequences). However, answering this question in general is a very costly algebraic task – we use an alternative approach which seems to be more efficient, but fails in disadvantageous situations.

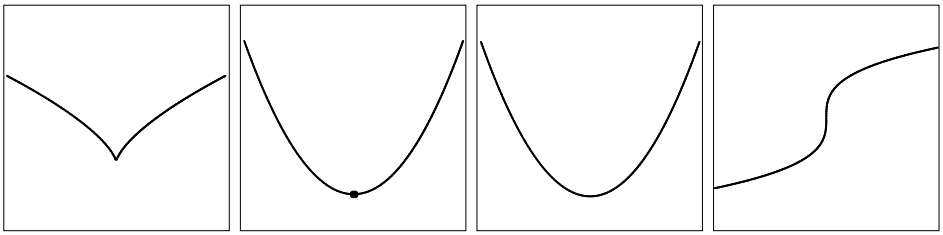

Figure 4.5.1: The curves $f_1 := x^2 - y^3$, $f_2 := (x^2 - y) \cdot (x^2 + y^2)$, $f_3 := x^2 - y$, and $f_4 := x - y^3$. f_1 and f_2 have singularities at the origin, whereas it is a non-critical point for f_3 and a vertical flex for f_4.

Assume that we know a rational expression for β in terms of α, i.e. that there exist integer polynomials p and q such that

$$\beta = \frac{p(\alpha)}{q(\alpha)}$$

To check whether (α, β) is singular is the same question as to ask whether

$$D_x f(\alpha, \frac{p(x)}{q(x)}) = 0 = D_y f(\alpha, \frac{p(x)}{q(x)}). \tag{4.5.1}$$

Defining

$$T_h(x) \quad := \quad q(x)^{\deg_y h} h(x, \frac{p(x)}{q(x)}),$$

(4.5.1) can be rewritten as

$$T_{D_x f}(\alpha) = 0 = T_{D_y f}(\alpha). \tag{4.5.2}$$

Since $T_{D_x f}$ and $T_{D_y f}$ are integer polynomials, we can check whether the algebraic number α (in interval representation) is a root using Algorithm 2.4.9.

But how can we find such a rational expression for β? If the polynomial f_α has only one multiple root over the complex numbers, we know from (2.5.6) that

$$\beta = -\frac{\mathrm{costha}_k(f_\alpha)}{k \cdot \mathrm{stha}_k(f_\alpha)} = -\frac{\mathrm{costha}_k(f)(\alpha)}{k \cdot \mathrm{stha}_k(f)(\alpha)}, \tag{4.5.3}$$

where k is the the degree of the greatest common divisor of f_α and f'_α. The idea is now as follows: We do not know whether f_α has indeed at most one multiple root over the complex numbers (and we do not want to check this in general because it is too expensive). But we can always form the rational expression

$$\rho(x) = -\frac{\mathrm{costha}_k(f)(x)}{k \cdot \mathrm{stha}_k(f)(x)}.$$

In a first step, we must verify that $\beta = \rho(\alpha)$, in other words that $\rho(\alpha)$ is indeed the y-coordinate of the candidate point. If it is, we can check the property given by (4.5.2) and set the `event` flag accordingly.

We formulate the single steps of `Generic Extension` for analysing the candidate. To check whether $\rho(\alpha) = \beta$, it proceeds in two steps. The first step checks whether $(\alpha, \rho(\alpha))$ is a point on the curve. With the same argument as above, this is equivalent to

$$T_f(\alpha) = 0.$$

If α is not a root of T_f, a failure is reported, because the rational expression does not yield the candidate. If it is a root of T_f, it is still not sure that $(\alpha, \rho(\alpha))$ is the candidate point. It might be some other covertical point on the curve which has accidently the y-coordinate $\rho(\alpha)$. To exclude this, `Generic Extension` approximates $\rho(\alpha)$ with interval arithmetic. Let J be the isolating interval for α.

1. Set $s_k := \mathrm{stha}_k(f), c_k := \mathrm{costha}_k(f)$.
2. Refine J until $s_k(J)$ does not contain zero.
3. Refine J until $\rho(J) = -\frac{c_k(J)}{k \cdot s_k(J)}$ overlaps with at most one isolating interval I of f_α.

After termination, it is certain that $\rho(\alpha)$ is the root of f_α inside I. If I is not the candidate interval, a failure is reported. If it is the candidate interval, $\rho(\alpha) = \beta$ is verified. Next, it is tested whether

$$T_{D_x f}(\alpha) = 0 = T_{D_y f}(\alpha)$$

is satisfied, and the event flag is set if and only if both equations hold.

We point out that these symbolic tests must only be performed in special occasions, namely in presence of vertical flexes which are not x-extreme, singularities with incidence numbers $(1, 1)$ (vertical cusps and isolated points on regular arcs),

or in case that f_α has imaginary multiple roots and a simple points was wrongly nominated as a candidate.

`Generic Extension` only reports a failure if $\rho(\alpha) \neq \beta$. But if the curve is $\mathbb{C}$-generic, each f_α has only one multiple complex point and ρ is always chosen correctly. Together with 4.3.4 it follows

Lemma 4.5.1. If the curve is $\mathbb{C}$-generic, `Generic Extension` succeeds.

This completes the proof of Proposition 4.1.2. Note that the algorithm can also succeed for curve which are not $\mathbb{C}$-generic (if they are at least $\mathbb{R}$-generic).

The expression ρ is also used in [GN02], but for a slightly different purpose: ρ is first used to ensure $\mathbb{C}$-genericity of the curve. Also, if the curve is $\mathbb{C}$-generic, then f_α can be made square free by dividing out $\gcd(f_\alpha, f'_\alpha) = (y - \rho(\alpha))^k$ (for a suitable k), and the roots of this square free polynomial can be computed. We do not check $\mathbb{C}$-genericity in our approach, and the roots of f_α can be isolated without making f_α square free beforehand.

4.6 A short summary

We briefly recapitulate: The goal of this chapter was the construction of the algorithm `Generic Extension`. We summarise its functioning:

Algorithm 4.6.1 (Generic Extension).
Input: Curve f, $\alpha_1, \ldots, \alpha_n$ x-values
Output: Vert-line objects for $\alpha_1, \ldots, \alpha_n$ (or a failure)

For each α_i, perform the following steps

1. Compute m, the number of points on the curve with x-coordinate α_i and k, the degree of $\gcd(f, D_y f)$ (Section 4.2).
2. (Try to) isolate the points supported by α_i, using `Bitstream m-k-Descartes` (Section 4.3).
3. Compute the incident numbers of each point supported by α_i (Section 4.4).
4. (Try to) check where the event point supported by α_i are (Section 4.5).

We believe that the substeps were sufficiently described in the corresponding sections and skip a more formal description.

How much symbolic calculation with the α_i's is necessary during `Generic Extension`? There are three different cases to consider:

- If α_i is a simple root of the resultant, the vert-line object can be constructed without any symbolic calculation with α_i.

- Otherwise, the number of real points supported by α_i is computed using the principal Sturm-Habicht coefficients. In many cases, this is the only necessary symbolic calculation with α_i.

- Some features of the curve cause additional symbolic calculations. Therefore, the principal and coprincipal Sturm-Habicht coefficients are necessary.

We remark that for $\mathbb{C}$-generic curves without singular points and vertical flexes, the extension phase is performed without any exact calculation of the α_i's.

The described algorithm reports a failure in some cases. As we want to give a complete solution, we must deal with these situations as well. In the next chapter, we will introduce an algorithm called `Nongeneric Extension` that creates the vert-line objects by using a linear change of coordinates. All substeps of `Generic Extension` also appear in `Nongeneric Extension` with minimal modifications.

Chapter 5

Extension Phase for Non-Generic Curves

In this chapter, we describe a second algorithm for the extension phase, called `Nongeneric Extension`, which was already outlined in Section 3.3. As input, it gets the input curve f and some values $\alpha_1, \ldots, \alpha_n$, containing the critical values of f. Additionally, it gets some shear factor $s \in \mathbb{Z}$ as input. The principal idea is that `Nongeneric Extension` tries to create the vert-line objects for f at the positions $\alpha_1, \ldots, \alpha_n$ by analysing the shearing of the curve f, called $\mathfrak{S}_s f$. This shearing arises from f by a linear change of coordinates. Applying a shear is a well-known technique in the analysis of algebraic objects, it also appears in [GK96, GN02, Wol02, EK+06]. We start with an investigation of the shear transformation in Section 5.1 and discuss the problems arising from changing the coordinate system in Section 5.2.

5.1 Shearing

We define the *shearing of a point* with shear factor $s \in \mathbb{R}$:

$$\mathfrak{S}_s(x, y) = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = (x + sy, y)$$

and for a point set

$$\mathfrak{S}_s M = \{\mathfrak{S}_s p \mid p \in M\}.$$

If it is clear what shear factor is used, we also write $\mathfrak{S}p$ instead of $\mathfrak{S}_s p$.

The corresponding *shearing of a curve* is defined as:

$$\mathfrak{S}_s f(x, y) := (f \circ \mathfrak{S}_{-s})(x, y) = f(x - sy, y)$$

Since $\mathfrak{S}_{-s}$ is inverse to $\mathfrak{S}_s$, it follows that $\mathfrak{S}f$ induces the point set of all sheared points on f:

$$f(p) = 0 \quad \Leftrightarrow \quad \mathfrak{S}f(\mathfrak{S}p) = 0. \tag{5.1.1}$$

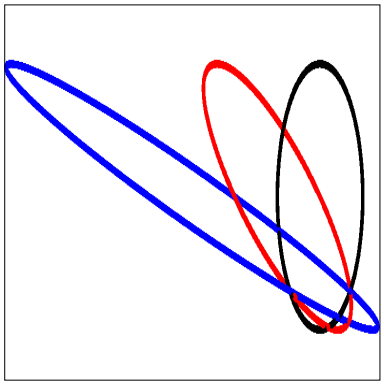

Figure 5.1.1: The curve $f = 2x^2 + (y-1)^2 - 2$, and the two shears $\mathfrak{S}_1 f$ and $\mathfrak{S}_3 f$

The shearing operator is defined as a mapping from the affine plane to itself (respectively, from the space of bivariate polynomials into itself), and the objects f and $\mathfrak{S}_s f$ are different polynomials for $s \neq 0$. There is also another viewpoint: Shearing might be considered as a basis change from the standard basis to the basis given by

$$\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} s \\ 1 \end{pmatrix} \right)$$

Thus shearing is nothing but choosing a different y-axis. Thus, instead of changing the objects like points and curves, the shearing can be considered to change the coordinate system, leaving all objects untouched.

Both descriptions can lead to quite different statements for the same idea. As an example, consider two covertical points p and q and apply the shearing operation with a non-zero shear factor. If you think that the basis changed, you can say now that p and q are not on the same parallel line to $x + sy = 0$, which is the y-axis in the sheared system. If you think that the map $\mathfrak{S}$ was applied, you might say that $\mathfrak{S}p$ and $\mathfrak{S}q$ are not covertical.

Both interpretations of shearing should be borne in mind – for the algorithmic description, it is better to use the idea of a map since it allows to consider $\mathfrak{S}f$ as an independent polynomial that can be analysed like any other curve. However, with the idea of a basis change, results are quite often clear by intuition. We use this intuition sometimes for explanations in this chapter.

We point out some easy but nevertheless important properties of shearing.

Theorem 5.1.1. Let $p \in \mathbb{C}^2$ and $s \in \mathbb{R}$ any shear factor:

1. The y-coordinate of p is invariant under shearing.
2. $f, g \in \mathbb{R}[x, y]$ have no common component iff $\mathfrak{S}f, \mathfrak{S}g$ have no common component.
3. f is square free iff $\mathfrak{S}f$ is square free.

4. p is intersection point of f and $D_y f$ iff $\mathfrak{S}p$ is intersection point of $\mathfrak{S}f$ and $\mathfrak{S}D_y f$.

5. $p \in \mathbb{R}^2$ iff $\mathfrak{S}p \in \mathbb{R}^2$.

6. p is singular for the curve f iff $\mathfrak{S}p$ is singular for $\mathfrak{S}f$

7. For $s \neq 0$: If p is a non-singular, critical point on f, then $\mathfrak{S}p$ is non-critical for $\mathfrak{S}f$.

Proof.

1. trivial

2. Follows from the fact that multiple components h of f and g cause multiple components $\mathfrak{S}h$ of $\mathfrak{S}f$ and $\mathfrak{S}g$ and vice versa.

3. Apply the previous statement with $g := D_y f$.

4. Follows immediately with (5.1.1).

5. trivial

6. For a point $(\alpha, \beta) \in \mathbb{R}^2$, we compute the gradient of $\mathfrak{S}_s f$ using the chain rule:

$$\begin{aligned}
\nabla \mathfrak{S}_s f(\alpha, \beta) &= \nabla (f \circ \mathfrak{S}_{-s})(\alpha, \beta) \\
&= (D_x f(\alpha - s\beta, \beta), D_y f(\alpha - s\beta, \beta)) \begin{pmatrix} 1 & -s \\ 0 & 1 \end{pmatrix} \\
&= (D_x f(\alpha - s\beta, \beta), -s \cdot D_x f(\alpha - s\beta, \beta) + D_y f(\alpha - s\beta, \beta))
\end{aligned}$$

   This implies

$$\nabla f(p) = 0 \Leftrightarrow \nabla \mathfrak{S} f(\mathfrak{S}p) = 0,$$

   as desired.

7. p is non-singular and critical, so $D_y f(p) = 0$ and $D_x f(p) \neq 0$. From the previous proof, we know that

$$(D_y \mathfrak{S} f)(\mathfrak{S}p) = \underbrace{-s \cdot f_x(p)}_{\neq 0} + \underbrace{f_y(p)}_{=0} \neq 0$$

   So, $\mathfrak{S}p$ is not critical for $\mathfrak{S}f$.

□

We turn to the following question: How likely is it that the curve $\mathfrak{S}_s f$ is $\mathbb{C}$-generic (Definition 4.1.1) for a random choice of s? The following theorem states that this is almost certain.

Theorem 5.1.2. There are only finitely many choices of s such that $\mathfrak{S}_s f$ is not $\mathbb{C}$-generic.

This property is very important for us: We will show that `Nongeneric Extension`, the extension algorithm described in this chapter, succeeds if $\mathfrak{S}_s(f)$ is $\mathbb{C}$-generic, hence it is very likely that the algorithm succeeds for a random choice of s.

The proof is quite long and needs mathematical concepts which are not necessary for the rest of this work. We decided to postpone the it to Appendix A as a bonus.

From now on, we assume $s \neq 0$ is a fixed shear factor.

5.2 Event points in a different coordinate system

Before we start with the definition of the algorithm, we discuss the main problem when working with a sheared curve. Using the algorithm from the last chapter, it would be possible to create the vert-line objects for $\mathfrak{S} f$ (at least, if the shear factor was chosen luckily and $\mathfrak{S} f$ is $\mathbb{C}$-generic). Unfortunately, event points depend on the coordinate system.

Example. Consider the curve $f = x + x^3 - y^2$. The origin is clearly an x-extreme point. Using the shear factor $s = -2$, we obtain:

$$g := \mathfrak{S}_{-2} f = 8y^3 + (12x - 1)y^2 + (6x^2 + 2)y + (x^3 + x)$$

Now, computing the resultant gives:

$$\operatorname{res}(g, D_y g, y) = 864x^4 - 32x^3 + 960x^2 + 3040x + 2016$$

and it is straightforward to check (with an computer algebra tool) that this polynomial has no real root. Thus g has no critical point.

This example shows that we lose information about the event points of f if we only consider $\mathfrak{S} f$. More precisely, we lose exactly the non-singular event points: Theorem 5.1.1 tells us that singular points of f are mapped to singular points of $\mathfrak{S} f$, whereas non-singular critical point always become non-critical if the curve is sheared with a nonzero shear factor. This is not surprising either: A non-singular critical point is only critical because its tangent happens to be parallel to the y-axis, but the partial derivative of a singular point vanishes in every direction. Additionally, some non-critical points of f become event points of $\mathfrak{S} f$, because the y-axis is chosen parallel to their tangent line. We introduce the following notation:

Definition 5.2.1.

- Let p be an event point of f. We call $\mathfrak{S} p$ a *sheared event point*, its x-coordinate *sheared event x-value*.
- A event point of $\mathfrak{S} f$ is called a *shear point*, its x-coordinate *shear x-value*.

The x-coordinates of shear values are roots of $R_{\text{sh}} := \text{res}(\mathfrak{S}f, D_y\mathfrak{S}f, y)$. We also want to detect the sheared event points of f in $\mathfrak{S}f$ because this allows to switch back to the original system later. The price we have to pay for this is a second resultant: Since all event x-values of f are roots of $\text{res}(f, D_yf, y)$, the sheared event x-values are roots of $R_{\text{ev}} := \text{res}(\mathfrak{S}f, \mathfrak{S}D_yf, y)$.

We will also use the following notation: For x-coordinates of critical points in the original system, we will use α or α_i with some index. When we talk about the sheared system, we will use σ or σ_i instead. As the y-coordinate does not change under shearing, we will always use β or β_i for it.

We restate the relation between sheared event point and shear points, and resultants in the following lemma.

Lemma 5.2.2.

1. If (σ, β) is a sheared event point, then $R_{\text{ev}} := \text{res}(\mathfrak{S}f, \mathfrak{S}D_yf, y)$ vanishes at σ.
2. If (σ, β) is a shear point, then $R_{\text{sh}} := \text{res}(\mathfrak{S}f, (D_y\mathfrak{S}f), y)$ vanishes at σ.
3. If (σ, β) is a singular point, then R_{ev} and R_{sh} both vanish at σ.

With the results from Theorem 5.1.1, it follows that both R_{ev}, and R_{sh} are non-zero polynomials if f is square free.

In summary, introducing a sheared curve complicates the situation both algebraically (more resultants) and geometrically (different types of event points). Still it can be used to create the vert-line objects in the original system. For that, we define the algorithm `Nongeneric Extension` which consists of two phases:

Algorithm 5.2.3 (Nongeneric Extension).
Input: Curve f, shear factor s, $\alpha_1, \ldots, \alpha_n$ roots of $\text{res}(f, D_yf, y)$
Output: Vert-line objects for $\alpha_1, \ldots, \alpha_n$

1. **Analysis phase:** Create a sequence of so-called *sheared-line* objects that contain information about sheared event points and shear points of $\mathfrak{S}_sf$ for certain x-coordinates.
2. **Backshear phase:** Use the sheared-lines to create vert-line objects for $\alpha_1, \ldots, \alpha_n$.

The analysis phase again divides into a projection phase and an extension phase. The projection phase is similar to the projection phase of f (Section 3.2), and the obtained values $\sigma_1, \ldots, \sigma_m$ contain all shear x-values and all sheared event x-values. The extension phase resembles the `Generic Extension`. It creates a simplified analogue of vert-line objects for $\mathfrak{S}f$, the sheared-line objects. This step can fail, and a failure is reported, but the algorithm succeeds at least if $\mathfrak{S}f$ is $\mathbb{C}$-generic. The main difference to the usual extension phase is that sheared event points are detected during the execution, although they are non-event points of $\mathfrak{S}f$. We describe this procedure in Section 5.3.

The backshear phase is a new method which allows to create the vert-line object in the original system using the sheared-line objects. The idea is that, as we have detected all sheared event points of f during the analysis, we can represent $\mathfrak{S}f$ combinatorially as a graph with sheared event points as nodes. For each sheared event point, we apply the inverse shearing to find out which α_i is the x-coordinate of the corresponding event point. Also, the behaviour of unbounded arcs of $\mathfrak{S}f$ is analysed when the inverse shearing is applied. With this information, the graph already contains enough information to compute most data of the vert-line objects. The last challenge is the computation of isolating intervals for f_{α_i}. We will introduce another instance of the `Generic Bitstream Descartes Algorithm` to compute them. The whole backshear phase is described in Section 5.4.

5.3 Analysis of $\mathfrak{S}f$ (analysis phase)

We start by specifying the output of the analysis. For each shear x-value and each sheared event x-values of $\mathfrak{S}f$, we build an object of the following form.

Definition 5.3.1. A *sheared-line* object is a triple

$$(\sigma, \texttt{number\_of\_points}, \texttt{points})$$

where σ denotes the x-coordinate of the sheared-line, `number_of_points` is the number of points on $\mathfrak{S}f$ supported by σ and `points` is a sequence of triples

$$(\texttt{approx}, \texttt{incidence\_numbers}, \texttt{sheared\_event}).$$

Each element of `points` stores information about one point over σ. `approx` is an isolating interval of the y-coordinate, `incidence_numbers` is a pair of integers denoting the number of arcs incident from the left and from the right. `sheared_event` is a flag which is set if and only if the point is a sheared event point.

`Nongeneric Extension` starts by computing the polynomial $\mathfrak{S}f$ and checks whether it is y-regular (i.e. whether the leading coefficient is a constant). It it is not, the presence of vertical asymptotes cannot be excluded and a failure is reported.

In the next step, the x-coordinates must be identified for which a sheared-line object is to be built. This is called sheared projection phase and we can define it directly with the results from the previous section:

Algorithm 5.3.2 (Sheared Projection Phase).
Input: Curves f, $\mathfrak{S}f$
Output: $\sigma_1, \ldots, \sigma_m$, containing all shear x-values and all sheared event x-values.

1. Compute the principal and coprincipal Sturm-Habicht coefficients of $\mathfrak{S}f$. This gives in particular $R_{\text{sh}} = \text{res}(\mathfrak{S}f, D_y\mathfrak{S}f, y)$, isolate its real roots.
2. Compute $R_{\text{ev}} = \text{res}(\mathfrak{S}f, \mathfrak{S}D_y f, y)$ and isolate its real roots.

3. Merge both root sets into one increasing sequence $\sigma_1, \ldots, \sigma_m$. Store for each element whether it is root of R_{ev} and root of R_{sh}.
4. Find rational values $r_0, \ldots, r_m$ such that $\sigma_i < r_i < \sigma_{i+1}$ (with $\sigma_0 = -\infty, \sigma_{m+1} = +\infty$).

We call an x-coordinate a *pure* R_{ev}-x-*value*, if it is a root of R_{ev}, but no root of R_{sh}. Similarly, a *pure* R_{sh}-x-*value* is a root of R_{sh} that is no root of R_{ev}. A *hybrid* x-*value* is a root of both R_{ev} and R_{sh}. The r_i's are called *intermediate* x-*values*.

The goal of the sheared extension phase is to create a sheared-line object for each σ_i. For that, the curve is first analysed at the intermediate x-values: For each r_i, the isolating intervals of $\mathfrak{S}f_{r_i}$ are computed. As r_i is a non-critical x-value of $\mathfrak{S}f$ and even a rational number, this can be done with the usual Descartes method. The algorithm will make use of these isolating intervals when it performs the extension step for pure R_{ev}-x-values and for hybrid x-values.

In the following three subsection we explain how the extension is done for pure R_{ev}-x-values, for pure R_{sh}-x-values and for hybrid x-values. It is possible that the extension fails for pure R_{sh}-x-values or for hybrid x-values, but it will satisfy the following property:

Lemma 5.3.3. If $\mathfrak{S}f$ is $\mathbb{C}$-generic, `Nongeneric Extension` successfully creates a sheared-line object for each σ_i.

5.3.1 Sheared extension phase for pure R_{sh}-x-values

Let σ be a pure R_{sh}-x-value. To fill the `number_of_points` field, the algorithm proceeds exactly as in Section 4.2. Note that the Sturm-Habicht sequence is only used for non-simple root of R_{sh} because otherwise the simplified method for counting the supported points applies. The isolating intervals are computed with the `Bitstream m-k-Descartes Algorithm` from Section 4.3. The incidence numbers are then obtained as in Section 4.4.

The last missing data is the `sheared_event` flag. This can be set to `false` immediately for each point because σ is not a root of R_{ev} and therefore, it cannot support any sheared event x-value.

5.3.2 Sheared extension phase for pure R_{ev}-x-values

Let σ be a pure R_{ev}-x-value. No critical point of $\mathfrak{S}f$ occurs, since R_{sh} does not vanish. Therefore, $\mathfrak{S}f_\sigma$ is square free, and the Bitstream Descartes method isolates the real roots. This step also gives the number of real roots supported by σ, so the Sturm-Habicht sequence is not needed here. The incidence numbers are clearly $(1, 1)$ for each point. It remains to set the `sheared_event` flags.

Let p be a point on f such that $\mathfrak{S}p$ is a point on $\mathfrak{S}f$ that is supported by σ. The question is whether or not p is an event point of f. Clearly, p is non-singular because otherwise σ would be a hybrid x-value. Theorem 2.2.25 tells us that p is

x-extreme if and only if $D_y f$ has different signs for the two incident arcs of p. For any point $q \in \mathbb{R}^2$, we have

$$D_y f(q) = D_y f(\mathfrak{S}^{-1}\mathfrak{S}q) = \mathfrak{S}D_y f(\mathfrak{S}q),$$

so p is x-extreme if and only if $\mathfrak{S}D_y f$ has different signs for the two arcs of $\mathfrak{S}f$ which are incident to $\mathfrak{S}p$. To find points on these arcs, we use the neighbouring intermediate lines of σ: Clearly, if $\mathfrak{S}p$ is the ith point of $\mathfrak{S}f_\sigma$, the ith points of the intermediate lines are on arcs incident to $\mathfrak{S}p$. Since we have chosen the r_i's as rational numbers, the y-coordinate of these arc points is given in interval representation, and the sign of $\mathfrak{S}D_y f$ at these point can be evaluated. Note that this can be done merely with interval arithmetic because $\mathfrak{S}D_y f$ cannot be zero at any point on the curve f with x-coordinate r_i.

Consequently, the algorithm advances i from 1 to n. p_i is the ith point on $\mathfrak{S}f$ over the left neighbouring intermediate x-value, q_i is the ith point over the right neighbouring intermediate x-value. The ith `sheared_event` flag is set to

$$\mathfrak{S}D_y f(p_i) \neq \mathfrak{S}D_y f(q_i)$$

Remark. This algorithm is capable to find several sheared event points over σ, so there is no genericity condition imposed at σ. Consequently, no failure is ever reported when analysing a pure R_{ev}-x-value.

5.3.3 Sheared extension phase for hybrid x-values

Let σ be a hybrid x-value. We start as in the case of a pure R_{sh}-x-value: The number of supported points, the isolating intervals and the incidence numbers are computed in the same way as in Chapter 4. It remains to find the `sheared_event` flags.

Recall that the `Bitstream m-k-Descartes Algorithm` returns not only the isolating intervals of $\mathfrak{S}f_\sigma$, but it also distinguishes one candidate interval with a candidate point inside. The candidate point is the only point which is possibly non-simple. As singular points always cause non-simple roots of $\mathfrak{S}f_\sigma$, the candidate interval is the only one which can contain a singular point.

But it can still happen that the other (simple) intervals contain sheared x-extreme points. Before we handle the candidate, we first consider the other points. As explained in the last section, it is enough to find points p, q on the two incident arcs of the point and to compare the signs of $\mathfrak{S}D_y f(p)$ and $\mathfrak{S}D_y f(q)$. The points p and q are again found by looking at the neighbouring intermediate x-values, the assignment is only slightly more complicated as in the case of pure R_{ev}-x-values.

Example. Consider a situation as in Figure 5.3.1. We start at the bottom. To set the `sheared_event` flag for the first point, the algorithm checks the signs of $\mathfrak{S}D_y f$ at the first arc at the left and the first arc at the right. For the flag of the second point, it considers the second arc at the left and at the right.

The third point is the candidate, this is handled later. The incidence numbers of the candidate are $(2, 4)$ in this example, so the next two arcs on the left and the next four arcs on the right must be skipped. For the fourth point, the algorithm must therefore check the fifth arc at the left and the seventh at right, and for the fifth point it must check the sixth arc at the left and the eighth arc at the right.

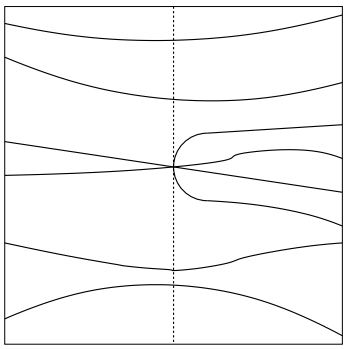

Figure 5.3.1: Example of a hybrid x-value.

We formulate the procedure for completeness:

1. Set $j := 1, k := 1$
2. For i from 1 to `number_of_points`
   i. if the ith point over σ is not the candidate point, compare the jth arc on the left and the kth arc on the right and set the *sheared_event* flag of the ith point over σ accordingly. Increase j and k by one.
   ii. If the ith point is the candidate, let (l, r) be its incidence numbers. Increase j by l, and k by r.

This procedure sets the `sheared_event` flags for the non-candidate points. There is only one remaining question: Whether the candidate is a sheared event point or not. We try to filter out easy cases with cheap arguments, and only if none of those simple techniques applies, we allow a more expensive calculation with the algebraic number σ.

For the easiest argument, we first look at the incidence numbers of the candidate. If their sum does not equal 2, this point is clearly singular and we are done (an example is in Figure 5.3.1). This filters out all candidates with not exactly two incident arcs and we only have to deal with candidates whose incidence numbers are $(1, 1)$, $(0, 2)$ or $(2, 0)$.

We present two further results which allow to set the `sheared_event` flag in special situations. If σ is a simple root of R_{sh}, Theorem 4.2.1 tells us that the candidate is a regular x-extreme point. Therefore its preimage is non-critical for f, and consequently the candidate is not a sheared event point. However, not all regular x-extreme points are filtered out by this method because imaginary multiple roots can push up the multiplicity of σ in R_{sh}.

The next lemma can be used to detect sheared event point in some cases. It is a simple rephrasing of Theorem 2.2.25

Lemma 5.3.4. Let p_1, p_2 denote two points on the two arcs of $\mathfrak{S}f$ incident to the candidate. If $\mathfrak{S}D_yf(p_1)$ and $\mathfrak{S}D_yf(p_2)$ have different signs, then the candidate is a sheared event point.

Proof. Assume that the signs are indeed different. If the candidate is singular, it is a sheared event point, and if it is not, the candidate must be a sheared x-extreme point according to Theorem 2.2.25. □

This lemma identifies all regular sheared x-extreme points as sheared event points, but there are examples of singular points where the signs at the two incident arcs do not change. Here is one typical situation:

Example. Consider a vertical cusp:

$$f = y^3 - x^2.$$

Passing to $\mathfrak{S}_{-1}f$ gives the equation

$$\mathfrak{S}_{-1}f = y^3 - (x+y)^2,$$

and the origin is still singular. We easily compute that $\mathfrak{S}_1 D_y f = D_y f = 3y^2$, and this is non-negative everywhere.

Finally, if the three criteria from above do not apply, we use an algebraic condition to detect singularities of $\mathfrak{S}f$. From the fact that $D_y\mathfrak{S}f$ and $\mathfrak{S}D_yf$ are partial derivatives of $\mathfrak{S}f$ in two linearly independent directions, it follows:

Proposition 5.3.5. A point p on $\mathfrak{S}f$ is singular if and only if $\mathfrak{S}D_yf(p) = D_y\mathfrak{S}f(p) = 0$.

To check that condition, we can proceed in exactly the same way as for the `event` flags in Section 4.5, using the rational expression

$$\rho(x) := -\frac{\mathrm{costha}_k(\mathfrak{S}f)(x)}{k \cdot \mathrm{stha}_k(\mathfrak{S}f)(x)}$$

for the y-coordinate. We summarise the complete procedure for determining the `sheared_event` flag of the candidate:

1. If the incidence numbers of the candidate are not in the set $\{(1,1),(0,2),(2,0)\}$, set the flag to `true`.
2. Otherwise: If σ is a simple root of R_{sh}, set the flag to `false`.
3. Otherwise: Find a point on each of the two incident arcs to the candidate, using the neighbouring intermediate x-values. If the sign of $\mathfrak{S}D_yf$ differs for those two points, set the flag to `true`.

4. Otherwise: Set
$$\rho(x) := -\frac{\mathrm{costha}_k(\mathfrak{S}f)(x)}{k \cdot \mathrm{stha}_k(\mathfrak{S}f)(x)}.$$
Check, whether $(\sigma, \rho(\sigma))$ is the candidate point. If it is not, report a failure. If it is, set the flag to `true`, if
$$\mathfrak{S}D_y f(\sigma, \rho(\sigma)) = 0 = D_y \mathfrak{S}f(\sigma, \rho(\sigma))$$
is satisfied, and to `false` otherwise.

This completes the description of the analysis phase in `Nongeneric Extension`. We remark that the algorithm indeed never reports a failure if $\mathfrak{S}f$ is $\mathbb{C}$-generic. A failure can only be reported during the execution of the `Bitstream m-k-Descartes` or during the symbolic check for the `sheared_event` flag, if $(\sigma, \rho(\sigma))$ does not represent the candidate. But both substeps always work if $\mathfrak{S}f$ is $\mathbb{C}$-generic.

5.4 From $\mathfrak{S}f$ to f (backshear phase)

In the last step, the sheared-line objects for the curve $\mathfrak{S}f$ were created. This section describes how this information can be used to build the vert-line objects of the curve f. We denote by $\alpha_1, \ldots, \alpha_n$ the real roots of $\mathrm{res}(f, D_y f, y)$, and by $\sigma_1, \ldots, \sigma_m$ the positions where the sheared-line objects are located (which means, the real roots of R_{ev} and R_{sh}).

It is conceptually easier to consider the curve $\mathfrak{S}f$ as a graph with its event points and some artificial nodes at $\pm\infty$ as nodes. The edges correspond to paths on the curve that connect two nodes. Formally, the edges correspond to the connected components of
$$\mathfrak{S}f \setminus \{p \mid p \text{ is a sheared event point of } \mathfrak{S}f\}.$$
We call the graph to create G_s. We distinguish two classes of nodes in G_s:
$$\begin{aligned} V_{\mathrm{fin}} &:= \{p \mid p \text{ is an sheared event point of } \mathfrak{S}f\} \\ V_{\mathrm{inf}} &:= \{+\infty_i \mid 1 \le i \le \#\{\text{arcs to } +\infty \text{ in } \mathfrak{S}f\}\} \\ &\quad \cup \{-\infty_i \mid 1 \le i \le \#\{\text{arcs to } -\infty \text{ in } \mathfrak{S}f\}\} \\ V &:= V_{\mathrm{fin}} \cup V_{\mathrm{inf}} \end{aligned}$$
V is the set of nodes of G_s. Note that regular shear points are not nodes of this graph. The construction of that graph G_s is discussed in Section 5.4.1. The edges of the graph correspond to paths in $\mathfrak{S}f$ connecting two sheared event points with no further sheared event point on the way. Therefore, the edges of G_s of $\mathfrak{S}f$ correspond to the arcs of f.

The graph G_s provides the topology of the curve f, as well as of $\mathfrak{S}f$ (shearing is a homeomorphic mapping). But to create the vert-line objects, geometric information is needed as well. Therefore, for each sheared event point of $\mathfrak{S}f$, the

x-coordinate of the corresponding event point of f is computed. This x-coordinate must be one of the α_i's since it must be an event x-value. We formally define a function:

$$\texttt{Index} : V \to \{-\infty, 1, \ldots, n, \infty\}$$

(n is the number of vert-line object to be constructed), which is defined for sheared event points as follows:

$$\text{For } \mathfrak{S}p \in V_{\text{fin}}, \texttt{Index}(\mathfrak{S}p) = i, \text{if } \alpha_i \text{ supports } p.$$

This means that the `Index` function computes the x-coordinate of the preimage of a sheared event point. We describe how to compute the `Index` of sheared event points in Section 5.4.2.

We also want to examine the behaviour of unbounded arcs of the original curve f. Therefore, we extend the `Index` function to nodes in V_{inf}. Each node $q \in V_{\text{inf}}$ is a symbolic endpoint of some unbounded arc A' of $\mathfrak{S}f$, and there is an unbounded arc A of f that was sheared to A'. There is only one meaningful return value of `Index` for such an unbounded arc:

$$\texttt{Index}(q) = \begin{cases} +\infty & \text{if } A \text{ is an arc to } +\infty \\ -\infty & \text{if } A \text{ is an arc to } -\infty \\ i & \text{if } A \text{ is an arc with vertical asymptote } x = \alpha_i \end{cases}$$

The `Index` function, applied on the endpoints of an edge, yields the x-range of the corresponding arc:

Proposition 5.4.1. Let $e = (v, w)$ be an edge of G_s and A be the corresponding arc of f. Let $i :=\texttt{Index}(v) < \texttt{Index}(w) =: j$. Then (α_i, α_j) is the x-range of A (with $\alpha_{-\infty} = -\infty$, $\alpha_{+\infty} = +\infty$).

For any arc converging to a vertical asymptote, we are also want to know whether it converges in direction $+\infty$ or $-\infty$. We define

$$\texttt{Sign} : \{v \in V_{\text{inf}} \mid \texttt{Index}(v) \neq \pm\infty\} \to \{\pm 1\}$$

as

$$\texttt{Sign}(q) = \pm 1, \text{if } q \text{ is an arc going to } \pm\infty$$

The computation of the `Index` and `Sign` function for unbounded arcs is the subject of Section 5.4.3.

Knowing the graph G_s and the functions `Index` and `Sign`, the vert-line objects of f are constructed in Section 5.4.4. Most data is easy to compute with one iteration over the graph. For instance, the number of points supported by α_i can be computed as follows:

$$\begin{aligned} & \#\{v \in V_{\text{fin}} \mid \texttt{Index}(v) = i\} \\ + \quad & \#\{(v, w) \in G_s \mid \texttt{Index}(v) < i < \texttt{Index}(w) \text{ or } \texttt{Index}(v) > i > \texttt{Index}(w)\}. \end{aligned}$$

The last remaining problem is that these points supported by α_i are not yet ordered with respect to their y-coordinate. We will derive another instance of `Generic Bitstream Descartes` that computes isolating intervals for the y-coordinates.

5.4.1 Building the graph

We describe how G_s can be constructed using the sheared-line objects from the analysis phase. In a first step, a slightly extended graph $\overline{G_s}$ is created: The shear points of $\mathfrak{S}f$ are included in the nodes set. Consequently, the edges of $\overline{G_s}$ are the same as for G_s, except that some edges are split into several pieces. The creation of $\overline{G_s}$ is done with a simple *sweep-line algorithm* [BK+97]. The idea is to consider a vertical line moving from $-\infty$ to $+\infty$ through the plane. It maintains a so-called *Y-structure*: At each position x, that structure contains edges supported by x in increasing order, and the structure is updated at sheared event x-values and shear x-values.

We will not give a formal definition of this sweep-line algorithm because its implementation is straightforward. After the creation of $\overline{G_s}$, search for a regular shear point in the node set, and find the two edges in $\overline{G_s}$ incident to it. Remove those two edges from the edge set and connect the two adjacent nodes by a new edge instead. Also, remove the shear point from the node set. Repeat until the node set is free of any regular shear point, the resulting graph is G_s.

It is possible to integrate this post-processing in the sweep-line algorithm, and our implementation does so. However, we will not discuss this in detail because this substep is not time-critical anyway, and it is longish to discuss all special cases.

5.4.2 Indexing event points

The next problem is to compute the value of the `Index` function for each node of the constructed graph. In this section, we start with the finite nodes, i.e. we explain how to compute the function for sheared event points.

We start by choosing intermediate values $q_i \in \mathbb{Q}, i = 0, \ldots, n$ with $q_{i-1} < \alpha_i < q_i$. These values partition the plane into $n + 2$ vertical stripes, which are labelled with $-\infty, 1, 2, \ldots, n, +\infty$. Let $\mathfrak{S}p$ be some sheared event point of $\mathfrak{S}f$, i.e. p is the corresponding event point of f. It is enough to determine the stripe in which p lies to assign $\texttt{Index}(\mathfrak{S}p)$. Let σ_i be the x-value of $\mathfrak{S}p$, β be its y-value. With shear factor s, the following equation holds:

$$\sigma_i - s \cdot \beta = \alpha_j$$

for some α_j. Since isolating (and refineable) intervals I_x for σ_i and I_y for β are known, interval arithmetic can be used: By computing $J := I_x - s \cdot I_y$, one obtains an interval containing α_j, and by refining I_x and I_y, J can be made arbitrarily small. In this case, it is only necessary to shrink J until it is completely contained in one interval $[q_{j-1}, q_j]$. It is clear then, that $\texttt{Index}(\mathfrak{S}p) = j$.

In geometric terms, I_x and I_y define a box B in the plane that contains $\mathfrak{S}p$. Refining the intervals means shrinking B. Now, J is the x-range of the object $\mathfrak{S}_{-s}B$, a parallelogram in the plane. The idea is to shrink the box until $\mathfrak{S}_{-s}B$ is contained in one of the stripes, and the point p must be also contained in this stripe, thus is supported by that α_i (see Figure 5.4.1 for an illustration).

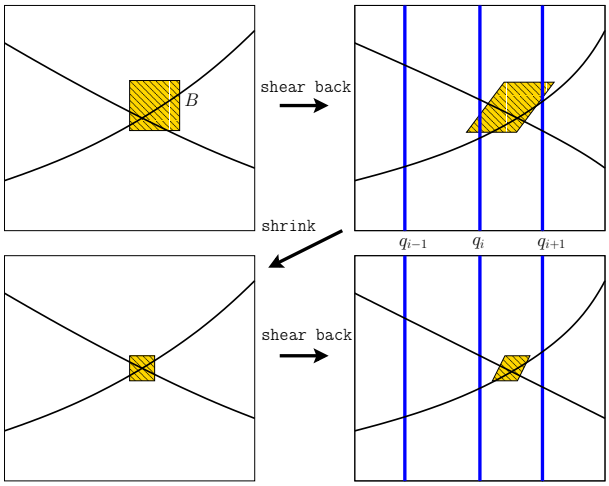

Figure 5.4.1: Above, the box of the event point intersects three stripes. After one refinement, it is completely contained in the stripe between q_i and q_{i+1} and hence, the index of this singularity is $i + 1$.

5.4.3 Indexing unbounded edges

We consider an unbounded arc of $\mathfrak{S}f$ now. Our goal is to find out the behaviour of the corresponding unbounded arc of the original curve, namely whether this arc is unbounded in x-direction, or whether it converges to a vertical asymptote. In the latter case, we also want to know whether it converges in direction $+\infty$ or $-\infty$.

Let us fix some unbounded arc $\mathfrak{S}A$ of $\mathfrak{S}f$ which goes to $+\infty$, the arcs to $-\infty$ are handled with the same idea. That unbounded arc $\mathfrak{S}A$ has a symbolic endpoint $+\infty_i \in V_{\text{inf}}$. If the backsheared arc A is unbounded in x-direction, it finally lies in one of the stripes $-\infty$ or $+\infty$. If it is an asymptotic arc to $x = \alpha_j$, it is finally contained in the stripe j. Thus, knowing where A "finally" lies would be enough to set $\texttt{Index}(\infty_i)$. The idea is to go "far" to the right in the sheared system, to the x-coordinate τ, and consider the point $\mathfrak{S}p$ on the arc $\mathfrak{S}A$ with x-value τ. If we are far enough on the right, the stripe of backsheared point p denotes the behaviour of the backsheared arc A.

We explain next why this intuition is right, and how to find such an x-value "far" enough at the right: Consider all intersection points of the curve f with a vertical line $x = q_i$. These are the places where an arc of f changes the stripe. As f is primitive, the number of intersections is finite, and consequently we can define a box B containing all those points. If this box is sheared, it becomes a parallelogram in the sheared system. We claim that being on the right of $\mathfrak{S}B$ means being far enough on the right (see also Figure 5.4.2 for an illustration).

Theorem 5.4.2. Let B be an axis-aligned box, containing all points on the curve f that are supported by some q_i. Let $\tau \in \mathbb{R}$ be greater than any x-coordinate

of $\mathfrak{S}B$, and let $\mathfrak{S}p$ be a point supported by τ on the unbounded arc of $\mathfrak{S}f$ with symbolic endpoint $+\infty_i$. Let $j \in \{-\infty, 1, \ldots, n, \infty\}$ be the stripe in which the backsheared point p lies. Then it holds that:

- $\texttt{Index}(\infty_i) = j$
- In case $j \in \{1, \ldots, n\}$: If p is above the box B, then $\texttt{Sign}(\infty_i) = 1$, otherwise $\texttt{Sign}(\infty_i) = -1$

Proof. Denote $\mathfrak{S}A$ the unbounded arc with symbolic endpoint ∞_i. Define the unbounded path from τ to $+\infty$ on $\mathfrak{S}A$ as follows:

$$\mathfrak{S}S := \{(a, b) \in \mathfrak{S}A \mid a \geq \tau\}$$

We consider the backsheared point set S, which is a path on the curve f. By definition of τ, S cannot enter the box B and consequently it cannot change the stripe. It follows that S is completely contained in the stripe j. If the arc converges to a vertical asymptote, and p is above the box, the whole of S must be above the box. The result follows. □

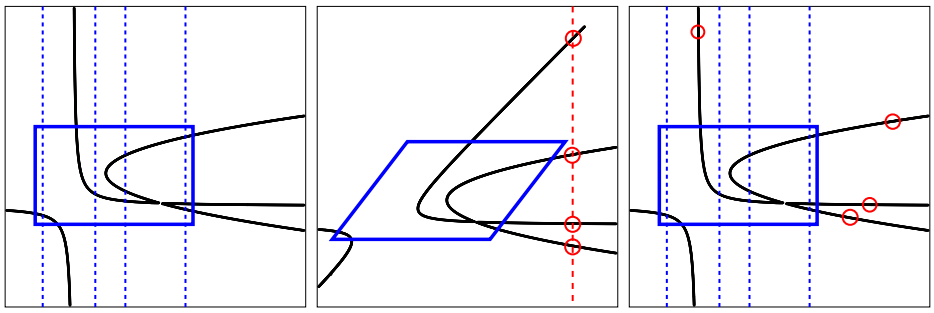

Figure 5.4.2: On the left, the original curve f. The blue box contains all points where the curve changes the stripe, the stripes are defined by the dashed lines. In the middle, the sheared curve, and the sheared box. The dashed red line on the right corresponds to τ in the text. On the right, we see the backsheared points of the sheared points supported by τ. We can see the behaviour of the unbounded arc by considering the location of those encircled points.

The box B can be found by applying a the Fujiwara bound S for each f_{q_i}. The complete procedure looks as follows:

1. Compute

$$M \quad := \quad \max\{S(f_{q_0}), \ldots, S(f_{q_n})\}$$

and define $B := [q_0, q_n] \times [-M, M]$.

2. Compute a rational value τ satisfying

$$\tau > \max\{|\sigma_i| \mid i \in \{1, \ldots, m\}\}$$

and

$$\tau > \max\{|q_0 \pm s \cdot M|, |q_n \pm s \cdot M|\}.$$

(The values $q_0 \pm s \cdot M, q_n \pm s \cdot M$ are the sheared corners of B.)
3. Isolate the real roots of $\mathfrak{S}f_\tau$, using the Descartes method.
4. For the ith point $\mathfrak{S}p_i$ supported by τ, find the stripe j in which the backsheared point p_i belongs, using interval arithmetic as in Section 5.4.2. In the case that $j \in \{1, \ldots, n\}$, decide whether p_i is above or below B.
5. Set $\texttt{Index}(+\infty_i) = j$ and, in the case that $j \in \{1, \ldots, n\}$, $\texttt{Sign}(+\infty_i) = 1$ iff p_i is above B.
6. Isolate the real roots of $\mathfrak{S}f_{-\tau}$, using the Descartes method, and repeat steps 4 and 5 to set $\texttt{Index}(-\infty_i)$ and $\texttt{Sign}(-\infty_i)$.

5.4.4 Construction of vert-line objects

Having constructed the graph G_s and knowing the values of `Index` and `Sign` for each node, we can finally create the vert-line objects of f. We recall once again which data the vert-lines consist of (Definition 3.1.1). They are seven-tuples

$$(\alpha, \texttt{vert\_comp}, \texttt{local\_degree}, \texttt{number\_of\_points}, \texttt{number\_of\_arcs\_lr}, \texttt{asym\_numbers}, \texttt{points}),$$

where `points` is a sequence of triples

$$(\texttt{approx}, \texttt{incidence\_numbers}, \texttt{event}).$$

Computing the degree of f_α The `local_degree` field is a special case among the seven entries of the vert-line, because it is the only field that does not need any information from the sheared curve, and it is the only field which is computed for all values α_i at once.

Let $f = \sum_{i=0}^{N} r_i y^i$ with $r_i \in \mathbb{Z}[x]$. Obviously, the degree of f_α is the maximal index d such that $r_d(\alpha)$ does not vanish. We define the following sequence of polynomials:

$$R_i := \begin{cases} \frac{r_i}{\gcd(r_i, r_i')} & \text{if } i = N \\ \gcd(R_{i+1}, r_i) & \text{if } i = 0, \ldots, N-1 \end{cases}.$$

For all α_i's which are not roots of R_N, it holds that $\deg f_{\alpha_i} = N$. For the roots α_i of R_N which are not root of R_{N-1}, it holds that $\deg f_{\alpha_i} = N - 1$ and so on. As the roots of each $\mathbb{R}_i$ form a subset of the α_j's, and each R_i is square free, we can immediately decide whether α_j is a root of R_i by sign evaluation at the boundaries of the isolating interval of α_j. We formulate the complete algorithm:

Algorithm 5.4.3.
Input: $f = \sum_{i=0}^{N} r_i y^i \in \mathbb{Z}[x, y]$, $\alpha_1, \ldots, \alpha_n$.
Output: The degree of all f_{α_i}

1. Set $d \leftarrow N$, $R \leftarrow \frac{r_N}{\gcd(r_N, r'_N)}$ and $A \leftarrow \{\alpha_1, \ldots, \alpha_n\}$.
2. While $A \neq \emptyset$ repeat
   i. For each $\alpha \in A$
      a. Check whether $R(\alpha) = 0$ by comparing the sign of R at the boundaries of the isolating interval of α.
      b. If $R(\alpha) \neq 0$, set `local_degree`$\leftarrow d$ for α's vert-line object and remove α from A.
   ii. Update $d \leftarrow d - 1$ and $R \leftarrow \gcd(R, r_d)$.

Note that A is the empty set before d reaches -1 because f is primitive.

Exploiting the edge set Let E be the edge set of G_s. We assume w.l.o.g. that for every edge $(p, q) \in E$, `Index`$(p) <$ `Index`(q). In fact, the indices cannot be equal since this would cause a vertical component in f, and if the index of q is smaller, we can swap the two entries.

The number of roots of f_{α_i} can be calculated with one iteration over the nodes and edges:

Lemma 5.4.4. For any α_i, $i = 1, \ldots, n$, the number of real roots supported by α_i is equal to

$$\#\{v \in V_{\text{fin}} \mid \texttt{Index}(v) = i\} + \#\{(v, w) \in E \mid \texttt{Index}(v) < i < \texttt{Index}(w)\}.$$

Proof. The left set counts the event points at α_i. All other points belong to arcs starting somewhere to the left hand side of α_i and ending to the right hand side of α_i. So they are counted by the second set. □

Similarly, the number of arcs converging to the vertical asymptote $x = \alpha_i$ can be computed:

Lemma 5.4.5. For any α_i, the number of arcs converging to the vertical asymptote $x = \alpha_i$ in direction $+\infty$ from the left is equal to

$$\#\{(v, w) \in E \mid w \in V_{\text{inf}},\ \texttt{Index}(w) = i,\ \texttt{Sign}(w) = +1\}.$$

The number of arcs converging to $x = \alpha_i$ in direction $+\infty$ from the right is equal to

$$\#\{(v, w) \in E \mid v \in V_{\text{inf}},\ \texttt{Index}(v) = i,\ \texttt{Sign}(v) = +1\}.$$

Similar results hold for the other asymptotic numbers. The number of arcs at $\alpha \pm \epsilon$ is also easy to compute:

Lemma 5.4.6. For any α_i, the number of arcs at the left of α_i is equal to

$$\#\{(v, w) \in E \mid \texttt{Index}(v) < i, \texttt{Index}(w) \geq i\}$$

These three lemmas are used to compute the fields `number_of_points`, `asym_numbers` and `number_of_arcs_lr` in the vert-lines, it remains to fill the `points` sequence.

Computing isolating intervals Fix one α_i and set

$$\mathrm{Ev} := \{v \in V_{\mathrm{fin}} \mid \texttt{Index}(v) = i\}, \quad e := \#\mathrm{Ev}$$

Let m denote the number of points on f supported by α_i. We know that there are e event points over α_i, and $m - e$ non-event points. Furthermore, we know that all non-event points have odd multiplicity. In other words, if an interval is isolating for a non-event root, the sign variation is odd.

For the e event points, we also know an approximation of their y-coordinate. This is because we know an isolating interval of the sheared event points over some σ_j, and the y-coordinate does not change under shearing. However, it is not sure that these intervals are also isolating for the original curve f. But still, these intervals allow to approximate the y-coordinate of the event points to any precision.

Let us summarise: We want to isolate the real roots of $g := f_{\alpha_i}$. What we know is:

- The number of real roots of g is m.
- A set Ev of e real roots can be approximated arbitrarily.
- All other roots have odd multiplicity.

We use another instance of `Generic Descartes` to isolate the real roots of g. We call an interval *marked*, if it contains an element of M, and *unmarked* otherwise. This can be checked by refining the y-coordinate of the elements in M. If the `leaves` sequence of the Descartes tree contains exactly e marked intervals, the roots in M are isolated against each other. The other roots have odd multiplicity, isolating intervals for them therefore have an odd number of sign variations by the Descartes' rule of signs. Consequently, we further refine until $m - e$ unmarked intervals with odd sign variation are found (compare Figure 5.4.3). We define the algorithm `Backshear-Descartes` as an instance of `Generic Descartes` and the following parameters:

- **Initial interval:** Apply the Fujiwara root bound S from Theorem 2.3.7 for g.
- **Termination condition:** `leaves` contains exactly e marked intervals and $m - e$ unmarked intervals with an odd number of sign variations.
- **Post-processing step:** Remove all unmarked intervals with an even number of sign variations from `leaves`.

For the implementation, we use the algorithm `Bitstream Backshear-Descartes`, that is an instance of `Generic Bitstream Descartes` with the same parameters as `Backshear-Descartes`. Note that the degree of f_{α_i}, which was already computed, is necessary to apply the Bitstream Descartes method for f_α (compare the remarks at the end of Section 3.6).

With the outcome of `Bitstream Backshear-Descartes`, the `approx` field is computed in the `point` sequence. For marked intervals, the `event` flag is set.

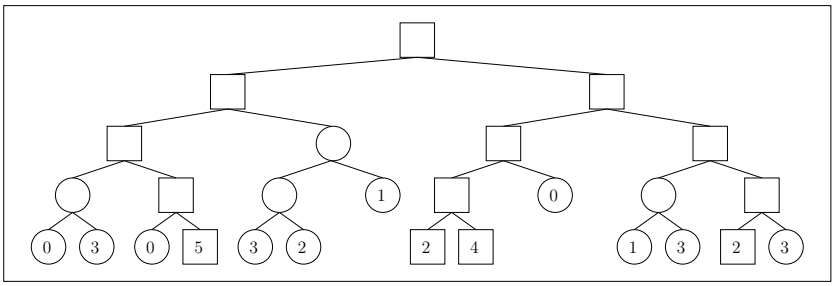

Figure 5.4.3: Let $e = 4$ and $m = 10$. Marked nodes are squares, unmarked circles. The algorithm terminates at this point (and throws away the unmarked interval with variation 2).

Exploiting the edge set – continued The only missing part is the number of incident arcs from the left and from the right for every point over α_i. Again, the incidence numbers for non-event points are $(1, 1)$. For an event point, let v denote its node in G_s. Then the number of arcs incident from the left is equal to

$$\#\{(w, v) \in E \mid w \in V\}$$

and from the right:

$$\#\{(v, w) \in E \mid w \in V\}$$

and finally, the vert-line objects are constructed.

Summary We recapitulate the single steps of the backshear phase: Constructing the graph G_s is a purely combinatorial problem and does not imply any calculation with algebraic numbers. The computation of the `Index` function for sheared event points is performed with interval arithmetic. For unbounded arc, the computation of τ only implies calculation with rational intermediate values q_i, and the computation of `Index` and `Sign` uses again interval arithmetic. The construction of the vert-line objects for f finally is done mainly by iterating over the edge set of G_s. The `local_degree` sequence is computed only by computing greatest common divisors of the coefficient polynomials of f, considered as polynomial in y. The isolating intervals for the vert-line objects are computed with the Bitstream Descartes method.

After all, the backshear phase computes the vert-line objects of f without any further symbolic calculation with the α_i's or σ_i's. Also, the backshear phase does not report a failure in any step. This implies that `Nongeneric Extension` always works, if the analysis phase succeeds. As we have seen, this happens at least if $\mathfrak{S}f$ is $\mathbb{C}$-generic.

Finally, we can argue that the whole extension phase terminates. Recall the two substeps (Section 3.3):

1. Try to apply `Generic Extension`. In case of success, return the vert-line objects.
2. In case of a failure, repeat choosing a shear factor s and apply `Nongeneric Extension` until it runs successfully. Return the vert-line objects.

By Theorem 5.1.2, there are only finitely many choices for s such that $\mathfrak{S}f$ is not $\mathbb{C}$-generic and these are the shear factors where `Nongeneric Extension` can possibly fail.

Chapter 6

AlciX – Algebraic Curves in EXACUS

After having introduced the complete algorithm for the analysis of a curve in the plane, we now turn to implementation questions. To summarise the complete algorithm, we repeat its high-level structure in Section 6.1. So far, we left out how basic operations should be implemented efficiently, and we will discuss several solution for that in Section 6.2. In Section 6.3, we figure out which choice of basic implementations is optimal with respect to the running time. Then, we analyse which operations are most expensive in the algorithm, we compare the running times with other approaches for topology computation and we give an idea how complex the input curves may become such that the analysis is still feasible.

6.1 Review of the algorithm

We summarise the whole algorithm for constructing the vert-line objects.

Algorithm 6.1.1.
Input: Square free polynomial $f \in \mathbb{Z}[x, y]$
Output: Vert-line objects (at least) for all critical x-values of f

1. Decompose $f = \text{cont}(f)\text{pp}(f)$.
2. Projection phase: Call Algorithm 3.2.5 on $\text{pp}(f)$.
3. Extension phase: Call Algorithm 3.3.1 on $\text{pp}(f)$, this means
   i. Try `Generic Extension` (Chapter 4)
   ii. On failure, try `Nongeneric Extension` with some random shear factor s (Chapter 5):
      a. Try the analysis phase for the sheared curve (Section 5.3).
         A. Sheared projection phase
         B. Sheared extension phase
      b. Backshear phase for the sheared curve (Section 5.4).
4. Set flags for vertical line components, considering $\text{cont}(f)$.

6.2 The implementation

This algorithm was implemented in C++ as part of the EXACUS library. Therefore, the module **AlciX** (**Al**gebraic **c**urves **i**n E**X**ACUS) was created and consists of about 8500 lines of code. We will also call our algorithm `AlciX` in this chapter.

We did not specify yet how several fundamental algorithms and data structures in `AlciX` are realised. For some of them, there are different possible choices and we implemented several alternatives to find the optimal one.

Fundamental data structures We need data types for the integers (coefficients of the input polynomial), and for the rationals (boundaries of isolating intervals for roots of polynomials). Both number types are independently provided by LEDA (as Integer and Rational) and CORE (as BigInt and BigRat).
Other data structures are derived from the integers and rationals in the natural way. For example, real algebraic numbers are represented as described in Section 2.4 using a polynomial and two rational numbers for the isolating interval.

Approximation of real algebraic numbers Whenever interval arithmetic is used to calculate with real algebraic numbers, the boundaries of the isolating interval are iteratively refined. Also, the Bitstream Descartes algorithm which is frequently used in the algorithm works with approximations of real algebraic numbers. Thus we need an efficient method for approximations.

In Section 2.4, we already discussed Abbott's method for the refinement of roots. Another possible solution is the repeated bisection of the isolating interval. Abbott's method promises quadratic convergence at some point, thus we expect a better performance compared to bisection. But it is not clear whether the computational overhead in each refinement step dominates the total costs in practice. The experiments will show that this is not the case and Abbott's method is more efficient in practice.

Interval arithmetic We use the interval data type from the boost library [BO] for calculations with intervals.

(Sub)resultants and Sturm-Habicht sequences The computation of the Sturm-Habicht sequence arises in the projection phase of the algorithm. Also, for each considered shear factor, the Sturm-Habicht sequence of $\mathfrak{S}f$ and the resultant R_{ev} must be computed. For the computation of the Sturm-Habicht sequence, we compute the corresponding subresultant sequence and change signs appropriately afterwards.

The normal way for the computation of the subresultant sequence is to perform the Euclidean algorithm but with certain scalar factors divided out in each iteration step (compare Theorem 2.5.5). Algorithms for this "classical" method have become textbook material [GCL92, §7.3],[Yap00, §3.5], [BPR03, §8.3.4]. Ducos [Du00]

combines two improvement on the classical method which further reduce the swell-up of intermediate coefficients. As stated in [LRS00], his solution seems to be among the best known solutions for the computation of subresultant sequences based on pseudo-division. We have implemented Ducos' algorithm.

A different approach is based on the following idea: The coefficients of each subresultant appear as minors of the *hybrid Bezout matrix* [DG04]. Thus the subresultant sequence can be computed only by evaluating determinants. [ADG04] shows that evaluating the determinant of the hybrid Bezout matrix with the *Samuelson-Berkowitz* method suffices to obtain the principal subresultant coefficients as they appear as a by-product of the computation. [Ke06] generalises that idea such that the first s coefficients of each subresultant are computed by evaluating the determinant of s matrices which are closely related to the hybrid Bezout matrix. In the curve analysis, we only need the first two coefficients of each subresultant (the principal and coprincipal subresultant coefficient), so only two determinant computations are enough. We also implemented this variant.

Ducos' algorithm performs $O(n^2)$ basic operations in $\mathbb{Z}[x]$ whereas the Bezout-based approach takes $O(n^4)$ such operations. However, the latter is free of division which turns out to be an advantage for domains with many parameters. The experiments will show that Ducos' algorithm is superior for bivariate polynomials.

Remark. We treat resultant and Sturm-Habicht computation as basically the same problem because the last element of the Sturm-Habicht sequence is the resultant. However, for the computation of the "extra" resultant R_{ev} in the sheared projection phase, it would also be possible to use a modular algorithm for the computation [Co71]. This should lead to a significant improvement of the practical performance for non-generic curves. We do not make use of the modular resultant algorithm in AlciX so far.

Greatest common divisor of univariate polynomials The computation of gcd's is contained in many substeps, for instance:

- The content is the gcd of the coefficients.
- For the isolation of the real roots of resultants (Algorithm 2.4.7).
- Comparison of real algebraic numbers (Algorithm 2.4.8) and sign evaluation of a polynomial at a real algebraic number (Algorithm 2.4.9).

As shown in Section 2.5, the gcd appears as the last non-vanishing subresultant in the subresultant sequence. Therefore, we can use the algorithm from the last paragraph for the computation of the gcd.

Another possibility is the usage of modular techniques: The gcd is computed modulo small prime numbers, and the results are combined using the Chinese remainder theorem, throwing away *unlucky* moduli. If the solution of the congruence divides both input polynomials, it is the greatest common divisor, otherwise more

prime numbers must be considered. That principle is also a well known technique covered by textbooks – see [GCL92, §7.4] or [GG99, §§6.4–6.7] for details. As the experiments in [GG99] show, it is very likely that the modular approach performs much better than any non-modular algorithm. Our code contains a modular method for computing gcds, using Victor Shoup's NTL library [NTL] as well as a non-modular method which is contained in EXACUS. The experiments will show that the modular method is indeed far superior.

Real root isolation For the isolation of integer polynomials, we use the square free factorisation together with the Descartes method in power basis, described in Algorithm 2.4.7. An implementation is contained in the NumeriX package of the EXACUS library.

For all variants of the Bitstream Descartes method (Bitstream Descartes, Bitstream m-k-Descartes, Bitstream Backshear-Descartes), Arno Eigenwillig has contributed a realisation of a Bitstream Descartes tree that can be explored individually by the application. This tree is the basis of the `Generic Bitstream Descartes Algorithm` and all its derivations.

6.3 Experimental results

6.3.1 Our test examples

We introduce a test suite of 8 curves. As we want to cover a large bandwidth of different algebraic properties, we try to include different features for each curve:

- rand_12_64 is a curve of total degree 12. Each coefficient is a randomly chosen integer of bit length 64.
- rand_8_512 is a curve of total degree 8. Each coefficient is a randomly chosen integer of bit length 512.
- 13_sings_9 is a curve of total degree 9 and maximal coefficient size 384. It contains 13 (non-covertical) singularities.
- ten_circles is the union of ten circles.
- covert_sings_8 is a curve of total degree 8 and maximal coefficient size about 3054. It contains 3 covertical singularities at two x-coordinates.
- L4_circles is the union of four circles with respect to the L_4 norm. The curve already appeared as f_5 in the appendix of [SW05].
- sing_asym_cont_7 is a curve of y-degree 7, total degree 15 and maximal coefficient size 668. It contains a singularity of high multiplicity, vertical asymptotes and vertical line components.

- cusps_and_flexes_9 is a curve of total degree 9 and maximal coefficient size 387, containing three vertical cusps and three vertical flexes.

Figure 6.3.1 displays graphs of the curves, plotted with MAPLE.

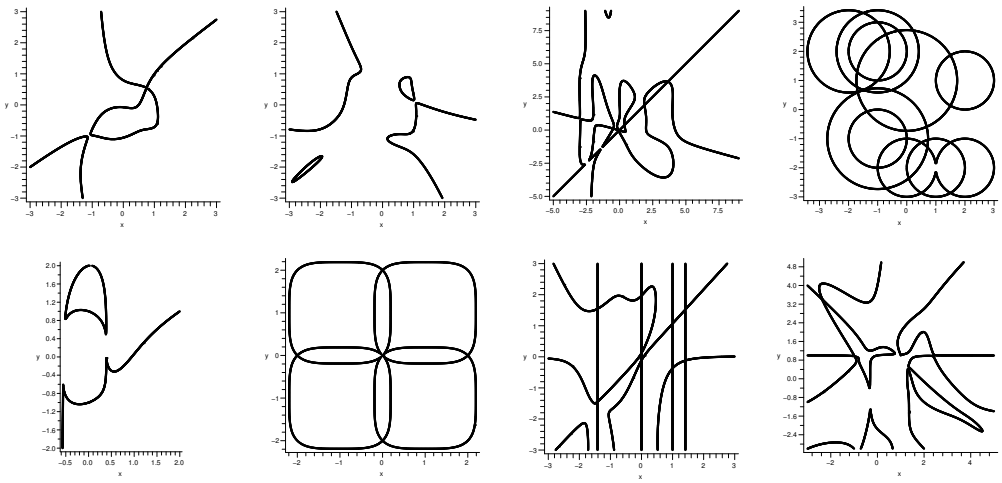

Figure 6.3.1: The curves of the test suite.

6.3.2 The optimal configuration and its alternatives

We will show that `AlciX` performs best when

- the CORE library is used to realise the integers and rationals,
- real algebraic numbers are refined using Abbott's quadratic refinement strategy,
- subresultants are computed using Ducos algorithm,
- gcds are computed with the modular algorithm.

The following table shows the running times for different configurations: For the first column, we used the configuration described above. For the second column, we replaced the CORE number types by LEDA number types (with LEDA version 4.4.1). In the third column, we changed the configuration from above by using bisection for refining real algebraic numbers. The fourth column gives the timings when Bezout matrices are used for subresultant computations, and the last column shows the effect of using a non-modular gcd algorithm. All timings are given in seconds.

Curve	Optimal	LEDA	Bisect	Bezout	Non-mod
rand_12_64	2.65	15.6	6.75	5.88	2.63
rand_8_512	2.27	13.8	3.89	3.76	2.12
13_sings_9	6.03	47.5	19.8	8.81	396
ten_circles	111	539	350	1313	1098
covert_sings_8	240	1598	264	329	8118
L4_circles	18.9	116	119	97.8	588
sing_asym_cont_7	135	530	187	291	8273
cusps_and_flexes_9	8.37	62.1	15.0	11.2	371

The table shows that deviating from the configuration proposed above worsens the the performance in a more or less dramatic way. From now, we will always use the optimal configuration.

6.3.3 Comparison with related work

We compare the algorithm with the "`insulate`" algorithm by Wolpert and Seidel [SW05] and "`top`" from Gonzalez-Vega and Necula [GN02]. We thank the authors of both to make their implementations available. We remark first that the significance of the presented timings is impaired by the following factors:

- The algorithms are implemented in different languages: Whereas `AlciX` runs in C++, the other two methods are only available as MAPLE implementations. However, this different platforms cannot explain the enormous improvements that we will notice in the experiments.

- Also, we point out that `AlciX` computes more than the other algorithms: `top` and `insulate` "only" return a graph with the topology of the curve, whereas `AlciX` also provides geometric output.

- Finally, `top` has an additional parameter that sets the initial precision of numerical calculations. Both a too low and too high initial precision impair the performance of the algorithm, and an appropriate precision is hard to find out a priori. We tested `top` with two initial precisions, 60 and 500.

We start with the 16 test curves from [GN02] (see Table 4 therein for the definition). Among these curves, some examples already appeared in [Ho96]. For `insulate` and `top`, MAPLE (version 10) was used on the same machine.

Curve	AlciX	insulate	top_{60}	top_{500}
f_1	0.27	4.56	1.40	3.22
f_2	0.01	0.14	0.12	0.21
f_3	0.01	0.16	0.14	0.22
f_4	0.04	0.18	0.09	0.14
f_5	0.04	0.14	0.09	0.12
f_6	0.23	1.78	0.54	1.27
f_7	0.04	0.27	0.17	0.42
f_8	0.04	1.66	0.24	0.47
f_9	0.17	1.66	0.48	1.01
f_{10}	0.17	0.66	0.33	0.79
f_{11}	0.09	1.87	1.05	3.22
f_{12}	0.54	19.36	25.55	37.33
f_{13}	0.01	0.11	0.10	0.16
f_{14}	0.30	2.76	1.15	3.32
f_{15}	0.01	0.12	0.02	0.03
f_{16}	0.10	0.46	0.20	0.33

We observe that `AlciX` analyses each curve under a second, and that it is faster than both other algorithm in each example. We point at the dramatical improvement for the curve f_{12}.

Next, we compare running times for the 5 examples from the appendix in [SW05].

Curve	AlciX	insulate	top_{60}	top_{500}
f_1	0.13	1.40	0.49	0.96
f_2	0.04	0.17	0.10	0.14
f_3	0.33	2.06	1.09	6.08
f_4	0.60	7.20	2.38	8.22
f_5	19.0	305	339	435

Again, our solution is faster, and for the complicated curve f_5, it improves on `insulate` by a factor 16.

Finally, we also tried to apply `insulate` and `top` to our test examples from Section 6.3.1.

Curve	AlciX	insulate	top_{60}	top_{500}
rand_12_64	2.65	19.5	63.7	252
rand_8_512	2.27	12.5	1648	8.22
13_sings_9	6.03	113	>4h	134
ten_circles	111	639	406	1226
covert_sings_8	240	990	>4h	>4h
L4_circles	14.9	305	339	error
sing_asym_cont_7	135	error	>4h	838
cusps_and_flexes_9	8.37	99.5	>4h	14.1

Again, `AlciX` is the best alternative. Note that `top` is competitive in some examples, but only because the initial precision is chosen appropriately.

6.3.4 A more detailed analysis of the algorithm

What are the expensive operations in `AlciX`? To examine this, we first classify three types of calculations:

- Calculations to identify critical x-values of f or of some sheared curve $\mathfrak{S}f$. This happens in the projection phase, and in the sheared projection phase of `Nongeneric Extension`. We call these calculations *Proj-operations*.
- Calculations for f_a, where a is some critical or intermediate x-values of f or $\mathfrak{S}f$. This happens in `Generic Extension`, and in the sheared extension phase of `Nongeneric Extension`. We call these calculations *Ext-operations*.
- The backshear phase of `Nongeneric Extension`.

The next table shows the timings for all Proj-calculations, Ext-calculations and for the backshear phase. If the backshear phase was not applied during the algorithm, the input curve clearly was sufficiently generic in the first place, and `Generic Extension` succeeded. The column titled "#vl" gives the number of created vert-lines. A large number of vert-line objects should increase the portion of the Ext-calculations on the total running time.

Curve	#vl	Total	Proj	Ext	Backshear
rand_12_64	8	2.65	1.80 (67%)	0.84 (32%)	-
rand_8_512	4	2.27	1.83 (81%)	0.43 (19%)	-
13_sings_9	35	6.03	3.60 (60%)	2.43 (40%)	-
ten_circles	33	111	77.2 (70%)	31.7 (29%)	0.75 (1%)
covert_sings_8	10	240	232 (96%)	6.81 (3%)	1.20 (1%)
L4_circles	15	18.9	14.9 (79%)	3.3 (17%)	0.51 (3%)
sing_asym_cont_7	13	135	125 (93%)	9.38 (7%)	0.52(0%)
cusps_and_flexes_9	20	8.37	3.51 (42%)	4.85(58%)	-

A first observation is that the backshear phase has a minor effect on the running time. This agrees with our remarks at the end of Chapter 5 where we pointed out that almost all substeps of the backshear procedure are either combinatorial or approximate. So we can concentrate on the first two classes.

The Proj-calculatiions tend to be more expensive in the examples, but there is also a counter-example in the last row, where the Ext-calculations take longer. This is not surprising: The input curves contains features like vertical flex points and vertical cusps, and in these cases, the algorithm has to check symbolically for the existence of event points, as they have incidence numbers $(1, 1)$ (Section 4.5).

We further decompose the Proj and Ext to identify the expensive steps of the algorithm. In Proj, we distinguish the calculations of resultants and Sturm-Habicht

sequences and the isolation of the real roots of resultants (this steps includes the square-free-factorisation and the Descartes method). The next table shows how the running time distributes over this two substeps:

Curve	Proj	Res. comp	Res. isol
rand_12_64	1.80	1.67 (93%)	0.134 (7%)
rand_8_512	1.83	1.71 (93%)	0.120 (7%)
13_sings_9	3.60	2.69 (75%)	0.91 (25%)
ten_circles	77.2	73.9 (96%)	1.88 (2%)
covert_sings_8	232	123 (53%)	108 (46%)
L4_circles	14.9	13.9 (93%)	0.76 (5%)
sing_asym_cont_7	125	109 (87%)	15.3 (12%)
cusps_and_flexes_9	3.51	2.76 (78%)	0.75 (21%)

We observe that the computation of the resultants and Sturm-Habicht sequences takes more time than finding their roots. This result is surprising, because the isolation step computes and divides out greatest common divisors of the resultant and its derivative, as explained in Section 3.2. We can see here how dramatically the modular gcd algorithm improves the performance. If we replace it by a the non-modular algorithm, the last column would have the entries: 0.118, 0.054, 391, 991, 7989, 568, 8156, 364.

For Ext, we measure the time for three suboperations: The computation of the numbers m (the number of real roots over some α) and k (the degree of the gcd of f_α and f'_α. Second, the running time of all `Bitstream m-k-Descartes` instances is measured. Finally, we measure the time to set the `event` flags of f and the `sheared_event` flags of $\mathfrak{S}f$. There are more operation involved in the extension phase, such as the computation at intermediate values or the computation of the incidence numbers. However, these operations are negligible with respect to the running time.

Curve	Ext	m-k-comp.	m-k-Descartes	ev. points
rand_12_64	0.84	0.16 (19%)	0.66 (79%)	-
rand_8_512	0.43	0.22 (51%)	0.21 (49%)	-
13_sings_9	2.43	1.40 (58%)	0.96 (40%)	-
ten_circles	31.7	12.35 (39%)	9.22 (29%)	10.1 (32%)
covert_sings_8	6.81	3.0 (44%)	3.55 (52%)	0.23 (4%)
L4_circles	3.3	0.94 (28 %)	1.66 (50%)	0.63 (19%)
sing_asym_cont_7	9.38	1.49 (16%)	3.25 (35%)	4.57 (49%)
cusps_and_flexes_9	4.85	0.87 (18%)	0.51 (11%)	4.83 (71%)

It seems that the first two steps are equally expensive, although there are examples where the Bitstream Descartes is dominant. If the curve contains "complicated" features like vertical cusps or vertical flex points, the symbolic event point check also contributes considerably to the total running time.

6.3.5 Feasibility bounds

What input curves are still practicable to analyse in a "reasonable" amount of time? There are two natural parameters that influence the running time:

- The (total) degree of the input polynomial
- The size of the coefficients of the input polynomial

We created 15 dense polynomial with total degree n random coefficients of bit length up to c. Then we analysed each of those curves. The following table shows the running times of the median curve and the worst-case curve. We also list the running time of the Proj-calculations.

	Median			worst case		
(n, c)	#vl	Total	Proj	#vl	Total	Proj
(11,256)	4	8.2	7.6	6	8.5	7.62
(12,256)	6	16.6	14.9	12	19.8	15.1
(13,256)	6	30.3	27.7	10	32.4	27.7
(14,256)	4	51.8	49.2	8	56.2	49.5
(15,256)	6	90.0	85.1	10	99.8	85.9
(16,256)	6	147.7	142.4	8	155.4	142.1
(17,256)	6	239.0	229.0	10	242.2	228.8
(18,256)	6	375.4	360.7	14	406.2	361.5

For increasing coefficient size, we get

	Median			worst case		
(n, c)	#vl	Total	Proj	#vl	Total	Proj
(11,256)	4	8.2	7.6	6	8.5	7.62
(11,512)	4	23.1	22.2	10	25.4	22.3
(11,1024)	6	66.9	63.1	14	71.7	63.7
(11,2048)	8	187.4	181.5	8	193.9	182.2
(11,4096)	4	564.4	546.0	8	582.4	547.4

However, these random curves seem not to be too realistic examples for real applications – for instance, the median curve for $(18, 256)$ only has 6 critical values, although the resultant is of degree $18 \cdot 17 = 306$ and there would be room for that many critical points for a curve of degree 18. This also explains the domination of the Proj-calculations for these instances. Looking at Figure 6.3.2, these curves do not look very complicated to analyse.

To generate more interesting curves, we proceed as follows: We fix a a positive integer n and create the polynomial of total degree n with indeterminates as coefficients. Now we choose a further positive even integer m and consider the grid

$$G := \{-\frac{m}{2} + 1, \ldots, \frac{m}{2}\} \times \{-\frac{m}{2} + 1, \ldots, \frac{m}{2}\}$$

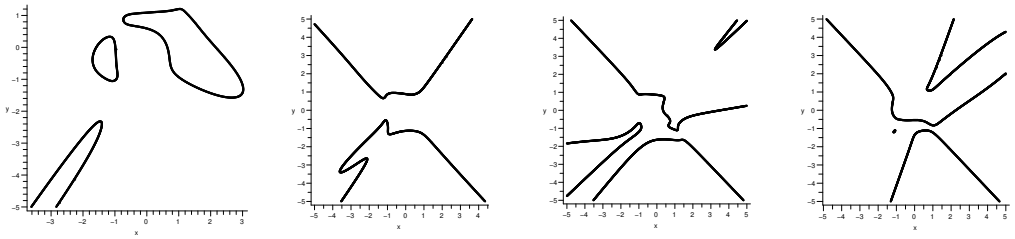

Figure 6.3.2: The graphs of the medians for $(n, c) = (12, 256), (14, 256), (16, 256)$ and $(18, 256)$.

We iteratively choose a point $p \in G$ and force the the curve to pass through this point. Each such point gives a linear equation in the coefficients, and we stop when the polynomial is uniquely determined (up to a scalar factor). By forcing the curve to pass these real points, we expect to include more real arcs of the curve and thus to produce more critical points.

The next tables shows the result of 15 randomly generated curve. The "bitsize" column shows the bit length of the greatest coefficient of the polynomial:

	Median				worst case			
(n, m)	#vl	bitsize	Total	Proj	#vl	bitsize	Total	Proj
(6,64)	14	275	0.27	0.08	14	295	0.39	0.09
(7,64)	18	376	0.92	0.39	20	422	1.31	0.48
(8,64)	18	541	3.37	1.94	18	559	4.20	2.10
(9,64)	18	695	11.33	7.26	32	732	14.81	7.65
(10,64)	28	868	35.84	22.82	38	1012	45.54	29.50
(11,64)	32	1101	99.81	70.19	44	1165	126.13	80.99
(12,64)	36	1388	269.64	204.11	52	1371	302.14	202.10

For increasing degree, also the size of the coefficients grows, as there are more points to interpolate. We can observe that the running time is about tripled if the degree is incremented. Also, the portion of the projection phase on the total running time is smaller, because there are much more critical values to consider. This can also be seen in Figure 6.3.3.

Remark. The running time of the analysis also depends on other factors. Local features as vertical cusps and vertical flexes impair the total running time since additional symbolic calculations are necessary, as seen in the example curve cusps_and_flexes_9. However, the influence of those features is very hard to measure systematically.

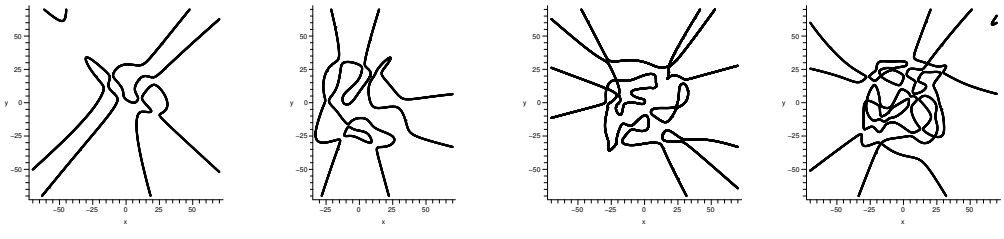

Figure 6.3.3: The graphs of the medians for $(n, m) = (6, 64), (8, 64), (10, 64)$ and $(12, 64)$.

Conclusion

This work contributes to the practical usability of the exact geometric analysis of real algebraic plane curves in three ways.

1.) It reduces the cost of the extension phase by replacing algebraic computations with more efficient approximate computations, without sacrificing exactness. For that, the Bitstream Descartes method is adapted to the case of multiple roots that appear naturally during the analysis. Sturm-Habicht sequences deal with the exact calculations concerning algebraic numbers that are still necessary. In most cases, a small number of such exact calculations suffices for the analysis.

2.) It performs the analysis in the predefined coordinate system. Coordinate changes can take place during the analysis, but the backshear phase brings the curve back into the original coordinate system.

3.) It further improves the performance by its lazy strategy of checking generic position: Depending on local features of the curve, different conditions are imposed on different x-coordinates. This leads to a fuzzy definition of genericity, which is less restrictive than $\mathbb{C}$-genericity, and can be verified more efficiently.

The described algorithm has been implemented as new module `AlciX` in the C++ library EXACUS. It outperforms the implementations of `top` [GN02] and `insulate` [SW05] in all examples. We therefore claim that `AlciX` reflects the state-of-the-art in the resultant-based analysis of algebraic curves. However, there is room for improvements, the usage of modular methods for the computation of resultants was already proposed in the text. Also, `AlciX` would benefit from further optimisations of the computation of subresultants and Sturm-Habicht polynomials since these computations are the dominant factor of the running time in most examples.

Appendix A

Multiple tangent lines

The subject of this appendix is a proof of Lemma 5.1.2:

Lemma. There are only finitely many choices of s such that $\mathfrak{S}_s f$ is not $\mathbb{C}$-generic.

Recall that $\mathbb{C}$-genericity imposes two conditions on f: f must be y-regular, i.e. the leading coefficient of f considered as univariate polynomial in y must be a constant, and each f_α must not have more than one multiple root. We show first that there are only finitely many shear factors that violate the first condition.

Lemma A.1.1. Let n be the total degree of f. There are at most n choices of s such that $\mathfrak{S}_s f$ is not y-regular.

Proof. The shearing transformation preserves degree, so also $\mathfrak{S}_s f$ has total degree n. Define

$$\overline{f} := \sum_{i=0}^{n} a_i x^i y^{n-i}$$

to be the sum of all terms in f with total degree n. A shear with s (considered as an indeterminate for the moment) yields

$$\overline{\mathfrak{S}_s f} = \sum_{i=0}^{n} a_i (x + sy)^i y^{n-i} =: \sum_{i=0}^{n} b_i(s) x^i y^{n-i}$$

where $b_i(s)$ depends on the shear factor. In particular, it holds that the coefficient of y^n is given by

$$b_0(s) = a_0 + sa_1 + s^2 a_2 + \ldots + s^n a_n$$

Not all a_i's are zero, so $b_0(s)$ is a non-zero polynomial in s. Now, if a concrete shear factor s is chosen with $b_0(s) \neq 0$, then $\mathfrak{S}_s f$ is y-regular. In other words, only a shear factor that is a root of $b_0(s)$ can spoil the y-regularity of $\mathfrak{S}_s f$. As there are at most n such roots, this proves the statement. □

We still have to prove that $\mathfrak{S}_s f$ violates the second property of Definition 4.1.1 for only finitely many choices of s. We reduce the statement to a geometric property of the curve f.

Definition A.1.2. A line $L(t) := (x(t), y(t)) \in \mathbb{C}^2$ is called *multiple critical line* of f if the polynomial $f(x(t), y(t)) \in \mathbb{C}[t]$ has more than one multiple root over $\mathbb{C}$. There are exactly three types of multiple critical lines:

1.) L passes through two singularities of f. We call these lines *multiple singular lines*.

2.) L passes exactly one singularity of f and is a tangent of f for some regular point. We call these lines *singular-tangent lines*.

3.) L passes no singularity of f and is tangent of f for (at least) two regular points. We call these lines *multiple tangent lines*.

Figure A.1.1 shows examples for each case. The following statement is obvious.

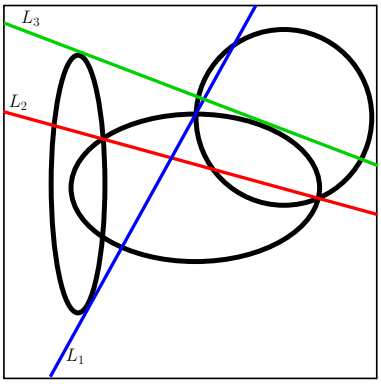

Figure A.1.1: A curve with three ellipsoid components. L_1 is a singular-tangent line, L_2 is multiple singular lines and L_3 is a multiple tangent line of the curve.

Proposition A.1.3. If L is a multiple critical line of f, then $\mathfrak{S}_s L$ is a multiple critical line of $\mathfrak{S}_s f$ (of the same type).

Look again at Figure A.1.1: If a shear factor is chosen such that the line L_2 becomes vertical, the sheared curve has two covertical singular points. If L_1 becomes vertical, the sheared curve has a singular point covertical to some x-extreme point of the sheared curve. If the line L_3 becomes vertical, two x-extreme points of the sheared curve are covertical. Thus, each multiple tangent line defines a bad shear factor:

Lemma A.1.4. Let f be a curve, and $s \neq 0$ be a shear factor such that $\mathfrak{S}_s f$ is y-regular. If $\mathfrak{S}_s f$ is not $\mathbb{C}$-generic, then f has a multiple critical line with slope $-\frac{1}{s}$.

Proof. If $\mathfrak{S}_s f$ is not $\mathbb{C}$-generic but y-regular, there must be some $\alpha \in \mathbb{R}$ such that $\mathfrak{S}_s f_\alpha$ has more than one multiple root. In other words, the line $L(t) = (\alpha, t)$ is a multiple critical line of $\mathfrak{S}_s f$. Shearing back, it follows that $(\mathfrak{S}_{-s} L)(y) = (\alpha - sy, y)$ is a multiple critical line of f. □

With that result, it is enough to prove that there are only finitiely many multiple critical lines. We argue that each of the three classes of multiple critical lines is finite. This is easy at least for the first class:

Lemma A.1.5. The number of multiple singular lines is finite.

Proof. The number of singularities is finite. For s singularities, there are at most $\binom{s}{2}$ connecting lines. □

For the remaining two classes, we use *dual spaces* ([BK81, pp.321+],[Wa50, II.3,V.8],[Gi96, 16.6]). We will only give a intuition about this concept, but we point out that this cannot replace a serious introduction in the theory of algebraic curves, as given in the named textbooks.

The basic idea is that dualising a point p in the plane gives a line p^* in the plane and dualising a line L gives a point L^*, such that 1.) $p^{**} = p, L^{**} = L$ and 2.) p lies on L if and only if L^* lies on p^*. For the formal construction, one has to embed the (affine) plane into the projective plane ([BK81, I.3],[Wa50, II.1],[Gi96, 9.1]) and define the dual of a point in the projective space.

The striking aspect of dualisation is that each statement about points, lines and their relation has an equivalent dual statement. We illustrate the principle with the easiest example. Consider the statement: *There is a line passing through any two points.* Let p_1 and p_2 lie on L. This means that L^* both lies on p_1^* and p_2^*, in other words L^* is an intersection point of the two lines. Because each line is the dual of some point and vice versa, we have the dual statement *Two lines always intersect in a point.* Note that we live in projective space here, so even parallel lines intersect.

Let f be an algebraic curve of degree n. For simplicity, we assume that f does not contain any line as a component. The *dual curve* f^* arises from f by dualising all tangents of f. We have only defined tangents for regular points, but one can extend the definition to singularities in an appropriate way (Figure A.1.2). The degree of the dual curve is at most $n(n-1)$, and it cannot contain lines as components.

We can prove next that the second class of multiple critical lines is finite.

Lemma A.1.6. The number of singular-tangent lines is finite.

Proof. Fix one singularity p. It is enough to show that only finitely many tangents of f pass through the point p. Each tangent passing through p corresponds to a point on the dual curve, and this dual point lies on the line p^*. In other words, tangent of f through p correspond to intersections of p^* with f^*. But a line can only intersect a curve in finitely many points. □

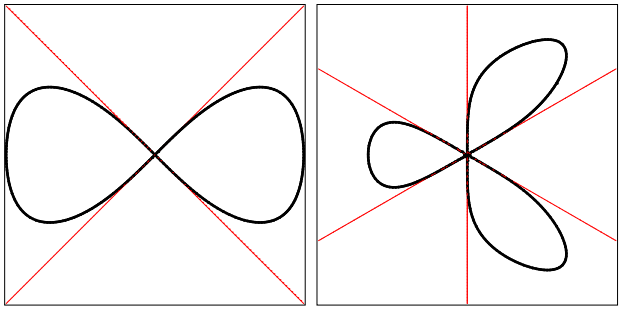

Figure A.1.2: Tangents at singular points.

There is only one unproven statement left:

Lemma A.1.7. The number of multiple tangent lines is finite.

In the nineteenth century, it was shown that the number of multiple tangent line is at most $\frac{1}{2}n(n-2)(n^2-9)$ ([Ja50, Cl64]). The result can also be found in [Ha77, §4, Exercise.2.3]. Since a curve has only finitely many singular points, it suffices to show:

Lemma A.1.8. Each multiple tangent line causes a distinct singular point of the dual curve.

We argue geometrically, see [Wa73, §§19–21] for an algebraic proof: Consider a multiple tangent line T which is tangential for two points p and q on the curve. We chose branches of f locally around p and q (Figure A.1.3). Each point on those branches has a unique tangent line, and dualising the set of tangents gives two branches of the dual curve f^*. Clearly, they intersect at least in the point T^*. If the dual branches are different, the point T^* is a self-intersection and we are done. It remains to argue that the branches cannot be equal. This follows from the general algebraic theory, but there is also an elementary argument which we present next. The idea is taken from a work of Horwitz [Ho89] which includes a proof that a function has only finitely many multiple tangent lines in a compact interval.

Definition A.1.9. Let p be a regular point on f with tangent T and let (p_i) be a sequence of regular points on f with tangents (T_i), all different from T. We define the *tangent intersection sequence of p and p_i* to be the sequence of intersection points of T_i with T.

Lemma A.1.10. If (p_i) converges to p, the tangent intersection sequence also converges to p.

The direct proof of [Ho89, Lemma 1] can be transfered. The dual statement says that, if a sequence (p_i) converges to p, the sequence of slopes of the secants

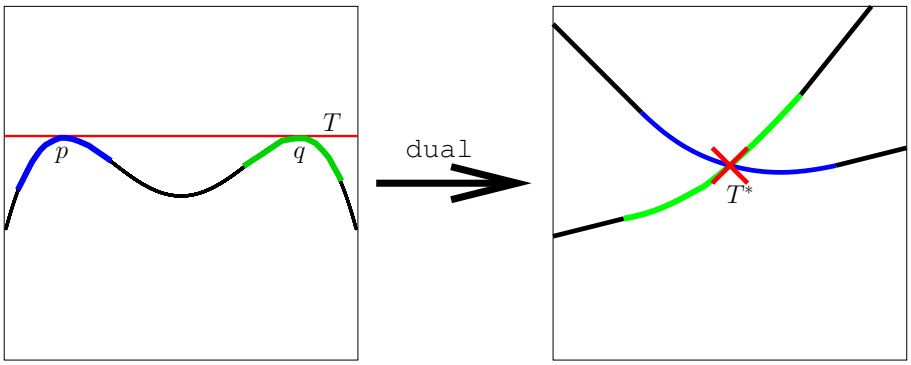

Figure A.1.3: The dual of the tangent T becomes a singular point if the dualised branch at p and the dualised branch at q are not equal.

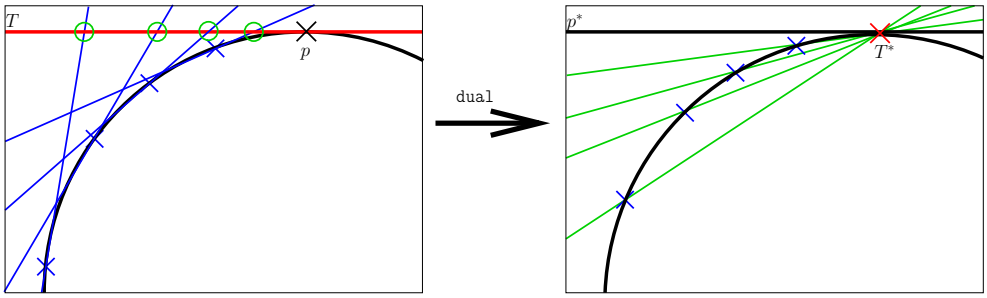

Figure A.1.4: On the left: Illustration of Lemma A.1.10. The encircled points define the tangent intersection sequence of p with the indicated sequence of points on the curve. On the right: The tangent intersection points correspond to secants through T^* of the dual curve.

through p_i and p converges to the slope of the tangent in p (Figure A.1.4). But this follows from the (analytical) definition of the tangent.

With the previous Lemma, we can prove that the branches for p and q cannot give the same dual branch. Assume for a contradiction that the dual branches for p and q are the same. We consider sequence of points on that dual branch, converging to T^*. This sequence induces two sequences of points on f: One sequence (p_i) converging to p, one sequence (q_i) converging to q, such that p_i and q_i share the same tangent T_i. Because f does not contain a line, all T_i's are different from T and Lemma A.1.10 says that the tangent intersection sequence of p and p_i converges to p, and the tangent intersection sequence of q and q_i converges to q. But both sequences are the same because the involved tangent lines are equal. Contradiction.

Bibliography

[Ab06] J. Abbott: *Quadratic Interval Refinement for Real Roots.* Unpublished, available at `http://www.dima.unige.it/~abbott/`.

[ACM84] D. Arnon, G. Collins, S. McCallum: "Cylindrical Algebraic Decomposition I+II". *SIAM Journal on Computing* **13** (1984) 865–889.

[ADG04] J. Abdeljaoued, G.M. Diaz-Toca, L. Gonzalez-Vega: "Minors of Bezout Matrices, Subresultants and the Parameterization of the Degree of the Polynomial Greatest Common Divisor". *Int. Journal of Computer Mathematics* **81** (2004) 1223–1238.

[Ap74] T. Apostol: *Mathematical Analysis*, Second Edition, Addison-Wesley, 1974.

[AS00] P. Agrawal, M. Sharir: "Arrangements and Their Applications". In: J.Sack, J.Urrutia (editors): *Handbook of Computational Geometry*, Elsevier, 2000, 49–119.

[Ba99] P. Batra: *Abschätzungen und Iterationsverfahren für Polynom-Nullstellen.* PhD thesis, Technical University Hamburg-Harburg, Germany, 1999.

[BE+05] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, N. Wolpert: "EXACUS: Efficient and exact algorithms for curves and surfaces". *13th Annual European Symposium on Algorithms* (ESA 2005), LNCS 3669, 155–166.

[BE+02] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, E. Schömer: "A Computational Basis for Conic Arcs and Boolean Operations on Conic Polygons". *European Symposium on Algorithms* (ESA 2002), LNCS 2461, 174–186.

[BH+05] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, N. Wolpert: "An Exact, Complete and Efficient Implementation for Computing Planar Maps of Quadric Intersection Curves". *21st Annual Symposium on Computational Geometry* (SCG 05) 99–106.

[BK81] E. Brieskorn, H. Knörrer: *Ebene algebraisch Kurven*, Birkhäuser, 1981 (German). English version: *Plane Algebraic Curves*, Birkhäuser, 1986.

[BK+97] M. de Berg, M. van Krefeld, M. Overmars, O. Schwarzkopf: *Computational Geometry: Algorithms and Applications*, Springer, 1997.

[BPR03] S. Basu, R. Pollack, M.-F. Roy: *Algorithms in Real Algebraic Geometry*, Springer, 2003.

[BO] The boost homepage: `http://www.boost.org/`.

[Bo01] S. Bosch: *Algebra*, 4. Auflage, Springer, 2001 (German).

[Br02] C. Brown: *Constructing Cylindrical Algebraic Decompositions of the Plane Quickly*. Unpublished, available at `http://www.cs.usna.edu/~wcbrown/`.

[BT71] J. Traub, W. Brown: "On Euclid's Algorithm and the Theory of Subresultants". *J. Ass. for Comp. Machinery* **18** (1971) 504–514.

[CA76] G. Collins, A. Akritas: "Polynomial Real Root Isolation Using Descartes' Rule of Signs". In: R.Jenks(editor): *Proceedings of the third ACM symposium on Symbolic and Algebraic Computation*, ACM press (1976) 272–275.

[CF+05] F. Cazals, J.-C. Faugère, M. Pouget, F. Rouillier: *Topologically Certified Approximation of Umbilics and Ridges on Polynomial Parametric Surfaces*. Rapport de recherce 5674, INRIA, Sophia Antipolis, 2005.

[CG] The CGAL homepage. `http://www.cgal.org/`.

[CJK02] G. Collins, J. Johnson, W. Krandick: "Interval Arithmetic in Cylindrical Algebraic Decomposition". *Journal of Symbolic Computation* **34** (2002) 143–155.

[Cl64] A. Clebsch: "Bemerkung zu Jacobis Beweis für die Anzahl der Doppeltangenten". *Journal für die reine und angewandte Methematik* **63** (1864) 186–188, available at `http://gdz.sub.uni-goettingen.de` (German).

[CL82] G. Collins, R. Loos: "Real Zeros of Polynomials". In: B.Buchberger, G.Collins, R.Loos (editors): *Computer Algebra: Symbolic and Algebraic Computation*, Springer, 1982, 83–94.

[CLO92] D. Cox, J. Little, D. O'Shea: *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, 1992.

[Co67] G. Collins: "Subresultants and Reduced Polynomial Remainder Sequences". *Journal of the Association for Computing Machinery* **14** (1967) 128–142.

[Co71] G. Collins: "The Calculation of Multivariate Polynomial Resultants". *Journal of the ACM* **18** (1971) 512-532

[Co75] G. Collins: "Quantifier Elimination For Real Closed Fields By Cylindrical Algebraic Decomposition". *Second GI Conference on Automata Theory and Formal Languages*, 1975, LNCS 33, 134–183. Reprinted in: B.F. Caviness and J.R. Johnson (editors): *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Springer, 1998, 85–121.

[DG04] G.M. Diaz-Toca, L. Gonzalez-Vega: "Various New Expressions for Subresultants and Their Applications". *Applicable Algebra in Engineering, Communication and Computing* **15** (2004) 233–266.

[Du00] L. Ducos: "Optimizations of the Subresultant Algorithm". *Journal of Pure and Applied Algebra* **145** (2000) 149–163.

[Eig06] A. Eigenwillig: "On Multiple Roots in Descartes' Rule and Their Distance to Roots of Higher Derivatives". To appear in: *Journal of Computational and Applied Mathematics*.

[ET05] I.Z. Emiris, E.P. Tsigaridas: "Real Solving of Bivariate Polynomial Systems". *8th International Workshop on Computer Algebra in Scientific Computing* (CASC 2005), LNCS 3718, 150–161.

[EK+05] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, N. Wolpert: "A Descartes Algorithm for Polynomials with Bit-Stream Coefficients". *8th International Workshop on Computer Algebra in Scientific Computing* (CASC 2005), LNCS 3718, 138–149.

[EK+06] A. Eigenwillig, L. Kettner, E. Schömer, N. Wolpert: "Exact, Efficient, and Complete Arrangement Computation for Cubic Curves". *Computational Geometry* **35** (2006) 36–73.

[ESY06] A. Eigenwillig, V. Sharma, C. Yap: "Almost Tight Recursion Tree Bounds for the Descartes Method". *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation* (ISSAC 2006) 71–78.

[EX] The EXACUS homepage. `http://www.mpi-inf.mpg.de/projects/EXACUS/`.

[FG+98] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, S. Schönherr: "The CGAL Kernel: A Basis for Geometric Computation". *ACM Workshop on Applied Computational Geometry* (WACG 1996), LNCS 1148, 191–202.

[Fo01] O. Forster: *Analysis 1*, 6. verbesserte Auflage, Vieweg, 2001 (German).

[Fo05] O. Forster: *Analysis 2*, 6. Auflage, Vieweg, 2005 (German).

[Fu16] M. Fujiwara: "Über die obere Schranke des absoluten Betrages der Wurzeln einer algebraischen Gleichung". *Tohoku Mathematical Journal* **10** (1916) 167–171 (German).

[GCL92] K. Geddes, S. Czapor, G. Labahn: *Algorithms for Computer Algebra*, Kluwer Academic Publishers, 1992.

[GG99] J. von zur Gathen, J. Gerhard: *Modern Computer Algebra*, Cambridge University Press, 1999.

[Gi96] C. Gibson: *Elementary Geometry of Algebraic Curves: An Undergraduate Introduction*, Cambridge Univesity Press, 1998.

[GK96] L. Gonzalez-Vega, M. El Kahoui: "An Improved Upper Complexity Bound for the Topology Computation of a Real Algebraic Plane Curve". *Journal of Complexity* **12** (1996) 527–544.

[GL03] J. von zur Gathen, T. Lücking: "Subresultants revisited". *Theoretical Computer Science* **297** (2003) 199–239.

[GL+98] L. Gonzalez-Vega, H. Lombardi, T. Recio, M.-F. Roy: "Sturm-Habicht Sequences, Determinants and Real Roots of Univariate Polynomials". In: B.F. Caviness and J.R. Johnson (editors): *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*, Springer, 1998, 300–316 .

[GN02] L. Gonzalez-Vega, I. Necula: "Efficient Topology Determination of Implicitly Defined Algebraic Plane Curves". *Computer Aided Geometric Design* **19** (2002) 719–743.

[Ha48] W. Habicht: "Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens". *Commentarii Mathematici Helvetici* **21** (1948) 99–116, available at `http://gdz.sub.uni-goettingen.de` (German).

[Ha04] D. Halperin: "Arrangements". In: J. Goodman, J. O'Rourke (editors): *Handbook of Discrete and Computational Geometry*, CRC Press, 2004.

[Ha77] R. Hartshorne: *Algebraic Geometry*, Springer, 1977.

[Ho89] A. Horwitz: "Reconstructing a Function From Its Set of Tangent Lines". *The American Mathematical Monthly* **96** (1989) 807–813.

[Ho96] H. Hong: "An Efficient Method for Analyzing the Topology of Plane Real Algebraic Curves". *Mathematics and Computers in Simulation* **42** (1996) 571–582.

[Ja50] C. Jacobi: "Beweis des Satzes daß eine Curve $n$ten Grades im Allgemeinen $\frac{1}{2}n(n-2)(n^2-9)$ Doppeltangenten hat". *Journal für die reine und angewandte Methematik* **40** (1850) 237–260, available at `http://gdz.sub.uni-goettingen.de` (German).

[JK97] J. Johnson, W. Krandick: "Polynomial Real Root Isolation Using Approximate Arithmetic". In: W. Küchlin (editor): *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation* (ISSAC 1997) 225–232.

[Ke06] M. Kerber: *Divison-Free Computation of Subresultants Using Bezout Matrices*. Technical Report MPI-I-2006-1-006, Saarbrücken 2006.

[KM06] W. Krandick, K. Mehlhorn: "New Bounds for the Descartes Method". *Journal of Symbolic Computation* **41** (2006) 49–66.

[Ko93] K. Königsberger: *Analysis 2*, Springer, 1993 (German).

[La68] S. Lang: *Analysis I*, Addison-Wesley, 1968.

[La93] S. Lang: *Algebra*, Third Edition, Addison-Wesley, 1993.

[LE] The LEDA homepage. `http://www.algorithmic-solutions.com/`.

[Lo82a] R. Loos: "Generalized polynomial remainder sequences". In: B. Buchberger, G. Collins, L. Loos (editors): *Computer Algebra – Symbolic and Algebraic Computation*, Springer, 1982, 115–138.

[Lo82b] R. Loos: "Computing in Algebraic Extensions". In: B. Buchberger, G. Collins, L. Loos (editors): *Computer Algebra – Symbolic and Algebraic Computation*, Springer, 1982, 173–188.

[LRS00] H. Lombardi, M.-F. Roy, M. Safey El Din: "New Structure Theorem for Subresultants". *Journal of Symbolic Computation* **29** (2000) 663-689.

[MS99] M. Mignotte, D. Stefanescu: *Polynomials: An Algorithmic Approach*, Springer, 1999.

[Mo79] R. Moore: *Methods and Applications of Interval Analysis*, SIAM, 1979.

[S05] S. Schmitt: "The Diamond Operator – Implementation of Exact Real Algebraic Numbers". *8th International Workshop on Computer Algebra in Scientific Computing* (CASC 2005), LNCS 3718, 355–366.

[Sy40] J. Sylvester: "A Method Of Determining By Mere Inspection the Derivatives From Two Equations of Any Degree". *Philosophical Magazine* **16** (1840) 132–135.

[MN00] K. Mehlhorn, S. Näher: *LEDA: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, 2000.

[NTL] The NTL homepage: `http://shoup.net/ntl/`.

[SW05] R. Seidel, N. Wolpert: "On the Exact Computation of the Topology of Real Algebraic Curves". *Proc. 21st Annual ACM Symposium on Computational Geometry* (SCG 2005) 107–115.

[Sl70] A. van der Sluis: "Upperbounds for Roots of Polynomials". *Journal of Numerical Mathematics* **15** (1970) 250–262, available at `http://gdz.sub.uni-goettingen.de`.

[Wa50] R. Walker: *Algebraic Curves*, Princeton University Press, 1950.

[Wa71] B. van der Waerden: *Algebra I*, 8. Auflage, Springer, 1971 (German).

[Wa73] B. van der Waerden: *Einführung in die algebraische Geometrie*, 2. Auflage, Springer, 1973.

[Wo96] J. Wolfart: *Einführung in die Zahlentheorie und Algebra*, Vieweg, 1996 (German).

[Wol02] N. Wolpert: *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. PhD Thesis, Saarland University, Germany (2002).

[Yap00] C. Yap: *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, 2000.

[YD95] C. Yap, T. Dubé: "'The Exact Geometry Paradigm". In: Du, Hwang (editors): *Computing in Euclidean Geometry* (Lecture notes series on computing 4), World Scientific, 1995, 452–492.

[Yun76] D. Yun: "On square-free Decomposition Algorithms". *Proceedings of the third ACM symposium on Symbolic and algebraic computation* (1976) 26–35.