Fast and Exact Geometric Analysis of Real Algebraic Plane Curves

Arno Eigenwillig
Max-Planck-Institut für Informatik
Saarbrücken, Germany
arno@mpi-inf.mpg.de

Michael Kerber
Max-Planck-Institut für Informatik
Saarbrücken, Germany
mkerber@mpi-inf.mpg.de

Nicola Wolpert
Hochschule für Technik
Stuttgart, Germany
nicola.wolpert@hft-stuttgart.de

ABSTRACT

An algorithm is presented for the geometric analysis of an algebraic curve $f(x, y) = 0$ in the real affine plane. It computes a cylindrical algebraic decomposition (CAD) of the plane, augmented with adjacency information. The adjacency information describes the curve's topology by a topologically equivalent planar graph. The numerical data in the CAD gives an embedding of the graph.

The algorithm is designed to provide the exact result for all inputs but to perform only few symbolic operations for the sake of efficiency. In particular, the roots of $f(\alpha, y)$ at a critical x-coordinate α are found with adaptive-precision arithmetic in all cases, using a variant of the Bitstream Descartes method (Eigenwillig et al., 2005). The algorithm may choose a generic coordinate system for parts of the analysis but provides its result in the original system.

The algorithm is implemented as C++ library `AlciX` in the EXACUS project. Running time comparisons with `top` by Gonzalez-Vega and Necula (2002), and with `cad2d` by Brown demonstrate its efficiency.

Categories and Subject Descriptors:
I.1.4 [**Symbolic and Algebraic Manipulation**]: Applications; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms*; G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*Polynomials, methods for*

General Terms: Algorithms, Performance

Keywords: Algebraic curves, cylindrical algebraic decomposition, topology computation, Descartes method, Sturm-Habicht sequence, exact geometric computation

1. INTRODUCTION

A bivariate polynomial f with integer coefficients defines an algebraic curve in the plane as its vanishing set. Our goal is to analyze the geometry of this curve f in the following sense: Imagine a vertical line ℓ moving from $x = -\infty$ to $x = +\infty$ through the plane. At each position, ℓ intersects f in finitely many points, as long as f has no vertical line as a component. While moving ℓ, the number of intersections can only change if f has a *critical point* (singularity or point with a vertical tangent line) at this x-coordinate or if an arc of f diverges at this x-coordinate, i.e., if ℓ is a vertical asymptote. We call all these positions *critical x-coordinates.* For each such critical x-coordinate α, our algorithm computes a *stack*, i.e., the y-coordinates of the points on the curve for that x-value (dashed lines at the right of Figure 1.1); these are the real roots of $f_\alpha(y) := f(\alpha, y)$. Furthermore, for each interval between critical x-coordinates, a stack is constructed for some sample point in that interval (dotted lines at the right of Figure 1.1), and it is computed how the points on neighboring stacks are connected to it. This extension from x-coordinates to stacks is called *lifting phase.* The ensemble of stacks gives a *cylindrical algebraic decomposition* (or *CAD*) of the plane (see [2], [7] for the general definition for n polynomials in $\mathbb{R}^d$) augmented with adjacency information. This result describes the topology of the curve as well as the position of critical points.

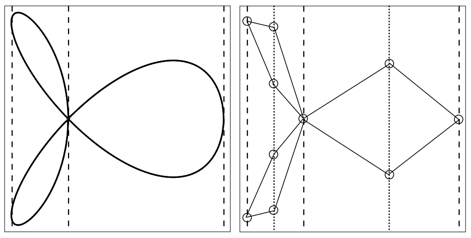
Figure 1.1: The critical values of an algebraic curve (left), and the result of the analysis (right)

There is a substantial body of previous work on computing a CAD; we mention the pioneering work led by Collins [7], [2], [3], work by Hong [17] and its references, and also the recent textbook [4]. A popular restriction of the problem is to compute just the topology of f, in the form of a topologically equivalent planar graph, see, e.g., [14], [15], [24]. This gives the freedom to change coordinates such as to escape from a degenerate position of f.

The literature agrees (e.g., [6], [9], [15], [17], [25]) that a pivotal source of efficiency is to avoid, as much as possible, exact arithmetic with the critical x-coordinates α in the lifting phase, because these are algebraic numbers, typically with a degree on the order of magnitude $\deg(f)^2$. Frequently ([6], [9], [17], [25]), verified fixed-precision arithmetic is used to solve easy cases fast, but exact arithmetic is still needed as a backup in case of failure. A different approach is exemplified by the work of Gonzalez-Vega and Necula [15]: use symbolic computations to remove a multiple root from $f(\alpha, y)$ and solve the remaining square free equation numerically; repeat at higher precision if the result looks suspicious. This works quite well in practice; however, [15] does not give a rigorous proof that a sufficient precision is selected for all inputs.

Our result. We present a solution that produces the *exact* result for *all* inputs, and it does so with an adaptive-precision numerical lifting phase. From a theoretical point of view, the uniform use of numerical lifting combined with the guarantee of an exact result is a pleasant novelty. From a practical point of view, the comprehensive use of approximate arithmetic makes our algorithm fast (see Section 7 for running time comparisons). The only "exact" information we need for a critical x-coordinate α is obtained from evaluating the signs of the principal Sturm-Habicht coefficients of f at α, see Section 2. This allows us to drive the Bitstream Descartes method for root isolation (presented in Section 3) properly in the presence of a multiple root.

During our algorithm, we may change from the original coordinate system into "sufficiently generic" coordinates. As a consequence, we will get adjacencies almost for free. This is a well-known trick for topology computation; see, e.g., [14], [15]. However, deciding genericity exactly requires symbolic computations. Unlike previous approaches, we abstain from a precise decision and do not rely on an initial genericity test. Instead, our analysis detects along the way whether a non-generic position poses a problem and only then triggers a change of coordinates. Thus we avoid the costly exact genericity test.

The benefits of coordinate changes are easy to use in algorithms for topology analysis [14] [15] [24], because topology is invariant under coordinate changes. However, we want an analysis in the *original* system. We exploit our analysis in the generic system to drive a numerical lifting in the original system and attain a CAD (with adjacencies) there. To our knowledge, this is new.

In the next two sections, we describe the two fundamental tools that we use. Our algorithm itself is introduced in Section 4. We report on our implementation and experimental comparisons in Section 7.

2. STURM-HABICHT SEQUENCES

Given a curve $f(x, y) = 0$ without vertical line components, our curve analysis needs to count the curve points on lines $x = \alpha$. These points are the distinct real roots of $f_\alpha(y) := f(\alpha, y)$. Sturm-Habicht sequences are a suitable tool to count them. For the reader's convenience, we repeat their definition and relevant properties. We use a simplified definition as in [15], and refer to [16] for proofs.

Definition 2.1. *Let $\mathbb{D}$ be any domain, $f \in \mathbb{D}[y]$ with $\deg f = n$, and $\delta_k := (-1)^{k(k+1)/2}$. For $k \in \{0, \ldots, n\}$, the kth Sturm-Habicht polynomial of f is defined as*

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

where $\mathrm{Sres}_k(f, f')$ is the kth subresultant of f and f'. We define $\mathrm{stha}_k(f)$, the kth principal Sturm-Habicht coefficient of f, as the coefficient of y^k in $\mathrm{StHa}_k(f)$.

The next two results are well-known from subresultant theory.

Theorem 2.2. *For any $f \in \mathbb{D}[y]$ of degree $n > 0$,*

$$\deg(\gcd(f, f')) = \min\{k \in \{0, \ldots, n-1\} \mid \mathrm{stha}_k(f) \neq 0\}.$$

Theorem 2.3 (Specialization property). *For a polynomial $f \in \mathbb{D}[x, y]$, let $(\mathrm{StHa}_i(f))_{i=0}^n$ be its Sturm-Habicht sequence w.r.t. y. Then, for any $\alpha \in \mathbb{D}$ with $\deg_y(f) = \deg(f_\alpha)$, $(\mathrm{StHa}_i(f)(\alpha))_{i=0}^n$ is the Sturm-Habicht sequence of the polynomial $f(\alpha, y)$.*

Sturm-Habicht sequences allow to count the number of distinct real roots in intervals (c, d). We only consider the interval $(-\infty, +\infty)$, for which use of the principal Sturm-Habicht coefficients suffices. For a sequence $I := (a_0, \ldots, a_n)$ of real numbers with $a_0 \neq 0$, we define the counting function $C(I) := \sum_{i=1}^s \epsilon_i$, where s is the number of subsequences of I of the form $(a, 0, \ldots, 0, b)$ with $a \neq 0$, $b \neq 0$, $k \geq 0$ intervening zeros, and

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

Theorem 2.4. *For $f \in \mathbb{R}[y]$ with $\deg f = n > 0$, we have*

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

Let us summarize: Given the curve $f(x, y) = 0$, we can consider the sequence $(\mathrm{stha}_n(f), \ldots, \mathrm{stha}_0(f))$ of its principal Sturm-Habicht coefficients w.r.t. y with a parameter x. Its last element $\mathrm{stha}_0(f)$ is the resultant $\mathrm{res}_y(f, \frac{\partial f}{\partial y})$ (up to sign). After specialization to a value $x = \alpha$ (subject to the degree condition from Theorem 2.3), the signs of $(\mathrm{stha}_n(f)(\alpha), \ldots, \mathrm{stha}_0(f)(\alpha))$ indicate both $k := \deg(f_\alpha, f'_\alpha)$ and $m := \#\{\beta \in \mathbb{R} \mid f_\alpha(\beta) = 0\}$.

3. BITSTREAM DESCARTES METHOD

The Descartes method [8] (see also [23], [13] and their references) computes *isolating intervals* for a square free univariate polynomial $g \in \mathbb{R}[t]$; that is, it assigns pairwise distinct enclosing intervals to the real roots of g. It is based on the following upper bound for the number of roots in an open interval (c, d). The *$[c, d]$-Bernstein basis* $(B_0^n[c, d], \ldots, B_n^n[c, d])$ of the vector space of polynomials of degree up to n is given by $B_i^n[c, d](t) = \binom{n}{i}(t-c)^i(d-t)^{n-i}/(d-c)^n$.

Theorem 3.1 (Descartes' rule of signs). *Let $g(t) = \sum b_i B_i^n[c, d](t) \in \mathbb{R}[t]$ have v sign variations in its coefficient sequence $(b_0, \ldots, b_n)$ and p roots in the interval (c, d), counted with multiplicities. Then $v \geq p$ and $v \equiv p \pmod 2$.*

The Descartes method maintains an interval queue, initially comprising a single interval that encloses all real roots. While the queue is non-empty, we remove its front element I and apply Descartes' rule to it. If $v = 0$, we know $p = 0$ and throw I away. If $v = 1$, we know $p = 1$ and output I as an isolating interval. If $v > 1$, we subdivide I and enqueue its two parts. For reasons that will become clear in Section 5, we insist that new intervals are enqueued at the back. If we think of the intervals inspected by the algorithm as a tree, this means the tree is traversed breadth first.

Although Descartes' rule appears weak compared to the exact root count offered by Sturm or Sturm-Habicht sequences, its simplicity makes the Descartes method very fast in practice, already when implemented with exact integer arithmetic [18] [19]. It can be accelerated further by replacing exact coefficients by approximations [9] [23], especially when the coefficients are not integers but more general algebraic numbers. However, as the method imposes a fixed grid of subdivision points, some inputs force it to exactly determine the sign of a vanishing coefficient, thus requiring to fall back from approximate to exact arithmetic [9, p. 152]. Eigenwillig et al. [12] have overcome this problem by randomizing the choice of subdivision points and controlling numerical precision adaptively. Their *Bitstream Descartes method* isolates the real roots of any square free real polynomial $g(t)$ whose coefficients are "bit-streams", i.e., arbitrary real numbers that are approximable to any positive absolute error but may not be known exactly. The necessary approximation precision is controlled automatically by the Bitstream Descartes method. The method comes with the rigorous guarantee that the resulting isolating intervals are valid for the exact polynomial $g(t)$ in all cases.

We show in Section 5 how to use the Bitstream Descartes method in the lifting phase of our algorithm, that is, for isolating the real roots of $f_\alpha(y) = f(\alpha, y)$, even if f_α has a multiple root. Its coefficients can be approximated to any desired accuracy by refining the isolating interval of α. For this, we use Abbott's Quadratic Interval Refinement [1].

4. CURVE ANALYSIS: OVERVIEW

The two preceding sections have introduced the two major tools for our algorithm. We now describe the algorithm itself. Let $f \in \mathbb{Z}[x, y]$ be the input polynomial that defines the algebraic curve. We restrict our exposition to the case that f is square free and has no vertical lines as components. Vertical lines can be divided out and added to the CAD after the analysis; we skip the details for brevity.

Our algorithm consists of two parts: The first part is a *direct method* of analysis by projection and lifting (see below) that always succeeds in a generic coordinate system but might reject a curve in non-generic coordinates. If the direct method rejects f, we change coordinates randomly until it succeeds. In that case, the result of an analysis in changed coordinates needs to be transformed back into the original coordinate system. This is achieved by the second part of our algorithm: a method of analysis in the original system that always succeeds, but depends on information from a successful direct analysis in a different coordinate system.

We take a look at the direct method first. We want to understand the curve's geometry at *critical x-coordinates.*

Definition 4.1. *A point $p \in \mathbb{R}^2$ on a curve f is* critical*, if $f(p) = 0 = \frac{\partial f}{\partial y}(p)$. If moreover $\frac{\partial f}{\partial x}(p) = 0$, p is called* singular*. Non-singular points are called* regular*. A* critical x-coordinate *is the x-coordinate of a critical point or of a vertical asymptote of f.*

It is well-known that critical x-coordinates of a curve are contained in the roots of the resultant $\operatorname{res}_y(f, \frac{\partial f}{\partial y})$. Our approach uses the standard strategy for the curve analysis (compare [2], [14], [15], [24]), which consists of two phases:

- *Projection phase*: Compute the real roots $\alpha_1 < \cdots < \alpha_n$ of the resultant, and rational sample points $\rho_0 < \alpha_1 < \rho_1 < \alpha_2 < \cdots < \alpha_n < \rho_n$ for each interval between roots.
- *Lifting phase*: (Try to) construct a *stack* for each α_i and each ρ_i, and compute how the points in neighboring stacks are connected.

For us, a stack is simply the increasing sequence of the y-coordinates for the curve points at some x-coordinate. We call the collection of stacks and their adjacencies a *CAD* for f (cf. the general CAD definition from Collins [2], [7]).

A stack is easily computed for a rational sample point ρ by using the Descartes method on the square free polynomial $f_\rho(y) = f(\rho, y)$. The interesting part is the stack construction over some root α of the resultant. Isolating the roots of $f_\alpha(y) = f(\alpha, y)$ efficiently is not easy because the coefficients of f_α are algebraic, and additionally because f_α is not square free. In Section 5, we present an efficient solution that is based on the following conditions.

Definition 4.2. *We call a curve $f \in \mathbb{Z}[x, y]$* generic*, if*

(G1) the leading term of f, considered as a polynomial in y, is a constant; and

(G2) for each real root α of $\operatorname{res}_y(f, \frac{\partial f}{\partial y})$, the polynomial $f_\alpha \in \mathbb{R}[y]$ has at most one multiple root in $\mathbb{C}$.

This notion of genericity also appears in [15], [14], [4, §11.6], and in a slightly more restrictive form in [24]. If (G1) is violated, we reject the curve immediately. Otherwise, we attempt lifting, as explained in Section 5. This succeeds if genericity holds. However, unlike previous approaches, it does not guarantee to reject the curve if genericity is violated; i.e., it does not *decide* genericity. This allows to avoid certain symbolic computations. Whenever lifting succeeds, it guarantees correctness of its result, whether f is generic or not.

Now we talk about the case that this direct method for lifting has not been successful. We transform the input curve by applying a *shear* (compare [24], [15], [14]). For a *shear factor* $s \in \mathbb{Z}$, the *sheared curve* is defined as

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

The curve $\operatorname{Sh}_s f$ is the image of f under the shear mapping $(x, y) \mapsto (x - sy, y)$ of the plane onto itself. Repeatedly choosing s at random and invoking the analysis eventually produces a CAD for $\operatorname{Sh}_s f$, since only finitely many shear factors make $\operatorname{Sh}_s f$ non-generic (see, e.g., [4, Prop. 11.23]). But the CAD for $\operatorname{Sh}_s f$ is not the one we are looking for; thus, we need the following additional phase.

- *Transformation phase*: Construct a CAD for f out of the CAD for $\operatorname{Sh}_s f$.

This is not trivial. The critical points of f do not correspond to those of $\operatorname{Sh}_s f$ (see Figure 6.1) and are therefore not covered by the stacks of $\operatorname{Sh}_s f$. Finding the image of all kinds of critical points of f on $\operatorname{Sh}_s f$ would necessitate further symbolic calculations in unfavorable cases. We will avoid this by considering a restricted class of critical points (Def. 6.1): An *event point* of the curve f is a point where f cannot be expressed locally as a continuous function in x, i.e., where the curve does not traverse from left to right.

Event points are always critical, but not vice versa: For instance, a vertical cusp (depicted on the left of Figure 6.2) is critical and even singular, but it is not an event point. Event points are detected more easily in the sheared system, and they contain the complete geometric information for the analysis. We refer to Section 6 for details.

5. DETAILS OF THE LIFTING PHASE

We describe how to produce the stack over a real root α of $\mathrm{res}_y(f, \frac{\partial f}{\partial y})$. Regarding genericity of f, condition (G1) of Def. 4.2 has already been checked, so f has no vertical asymptotes. (G2) has not yet been checked; we may reject f if it turns out that (G2) is violated.

We isolate the real roots of $f_\alpha(y) = f(\alpha, y)$ using a variant of the Bitstream Descartes method (Section 3). We want to avoid the initial step of making f_α square-free. Instead, we use the principal Sturm-Habicht coefficients of f, specialized to $x = \alpha$, and obtain m, the number of distinct real roots of f_α, and k, the degree of $\gcd(f_\alpha, f'_\alpha)$; see Section 2. Then we apply the Bitstream Descartes method directly on f_α. Recall from Section 3 that the method maintains an interval queue, and this queue will never become empty in case of multiple roots. Therefore, we interrupt the execution if one of the two following termination conditions is satisfied. Either, $m - 1$ simple roots of f_α are found, and there is only one more interval in the interval queue: this is the success case of the algorithm, we have found exactly m isolating intervals. Or, no interval in the queue has more than k sign variations: the curve is rejected in this case. We call this the *m-k-Descartes algorithm* to point out that it needs knowledge about m and k.

LEMMA 5.1. *m-k-Descartes terminates for any polynomial f_α.*

PROOF. By the termination proof for the square-free case (cf. [21]), all intervals free of multiple roots count 0 or 1 sign variations eventually. If f_α has at most one multiple root, this implies that the first condition is eventually satisfied. If f_α has more than one multiple root over $\mathbb{C}$, then each root has multiplicity at most k, because each multiple root contributes at least one to the degree k. It has been shown [11] that any sufficiently small interval containing an r-fold root of f_α counts exactly r sign variations. Therefore, from that point onwards, all intervals count at most k. □

LEMMA 5.2. *If the curve f is generic, m-k-Descartes succeeds for any f_α.*

PROOF. It is enough to show that the second termination condition is never satisfied for f_α. By the definition of genericity, f_α has at most one multiple root over $\mathbb{C}$, so $\gcd(f_\alpha, f'_\alpha) = (x-\beta)^k$ for some $\beta \in \mathbb{R}$. Thus, β is a $(k+1)$-fold root of f_α, and an interval containing β will always count at least $k+1$ sign variations. □

We point out two further properties of the m-k-Descartes algorithm. First, it can be also successful for polynomials with more than one multiple root: If f_α has at most one real multiple root and further imaginary ones, either termination conditions may be satisfied after suitable subdivision of the initial interval, and one cannot easily predict which one is satisfied first.

Second, on success, the sign variations counted for the last remaining interval in the queue only give an upper bound of the root's multiplicity with the correct parity (Theorem 3.1). For odd multiplicities, it is not certified that the root inside is in fact multiple. Translated to our geometric problem, we cannot guarantee that such a point over α is critical. We call such roots (points) *candidates roots (points)*. All non-candidate roots are non-critical. Observe that root isolation with Sturm's method instead of the Descartes method would not provide any information about roots being simple or multiple.

If lifting succeeds for all α, we have computed stacks for each critical x-coordinate. It remains to find the adjacencies between the points at α with the left and right neighboring stacks. Let ρ^- and ρ^+ be the x-coordinates of the sample points for the neighboring stacks. We say that a point p on α is adjacent to a point p^+ (or p^-) on ρ^+ (or ρ^-) if there is an x-monotone curve segment that joins p and p^+ (or p^-, respectively). A non-critical point at α must have exactly one adjacent neighbor at the left and at the right by the implicit function theorem.

The output of the m-k-Descartes algorithm, together with our genericity conditions, allows to compute adjacencies in a purely combinatorial way, as explained next. The same method was already used in [15], and it is more efficient than putting boxes around critical points and performing root isolation at the box boundaries, as other approaches do [3] [17] [24].

Assume that f_α has $a+b+1$ roots, where a roots are above the candidate point p, and b roots below. Further assume that there are m^+ (or m^-) points on the stack of ρ^+ (or ρ^-, respectively). The curve does not have a vertical asymptote at α, so the a points above p are adjacent to the a highest points at ρ^- and at ρ^+. The analogous statement is true for the b points below p. The remaining $m^- - a - b$ arcs on the left and the remaining $m^+ - a - b$ on the right are adjacent to p. An example is depicted in Figure 5.1. Note that both the presence of vertical asymptotes and more than one candidate over α would spoil this simple argument.

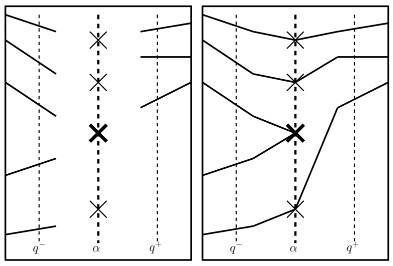

Figure 5.1: Example for the adjacency algorithm.

It might appear unsatisfactory not to have more validated geometric information about the candidate. Optionally, one can decide whether the candidate is singular with some extra symbolic computations: For generic curves, the candidate's y-coordinate has a rational expression in terms of the x-coordinate [14] [15]. With that, one can check whether the partial derivatives vanish or not [20, §5.4].

We remark an optimization for the lifting step that can be applied quite often. If α is a simple root of the resultant, there is exactly one critical point at α, and this point is a regular *x-extreme point*, i.e., a point having greatest (or smallest) x-coordinate among all curve points in some neighborhood [20, Thm. 4.2.1]. This case was already treated by Brown [6, §3.2]. The number of roots at the two neighboring stacks of α differ by exactly two (the two points adjacent to the x-extreme point), and the number of roots over α is the mean of these values. Moreover, since there is only one critical point, the m-k-Descartes algorithm will always be successful. In other words, the stack for such an α is constructed without using the Sturm-Habicht sequence, or other symbolic calculation with α.

6. DETAILS OF THE TRANSFORMATION PHASE

Let us now treat the case that the curve f was rejected because of non-genericity, but $\mathrm{Sh}_s f = f(x + sy, y)$ has successfully been analyzed by the method described above. For brevity, we omit s and write $\mathrm{Sh}f$. For a more compact description, let us assume temporarily that f is a bounded subset of $\mathbb{R}^2$, postponing the treatment of unbounded curves to §6.3.

Our goal is to construct a CAD for f, the original curve. The following class of points on f will be of importance.

Definition 6.1. *A point p on f is an* event point*, if, locally around p, the curve f is not the graph of a continuous function $y = \varphi(x)$.*

Geometrically, event points are either self-intersections of the curve, isolated points of the curve, or x-extreme points. In terms of a CAD, event points are precisely those points that do not have exactly one adjacent point in both the left and right neighboring stack. Event points are always critical, but not vice versa.

The event points of the curve induce the *event graph* for the curve, a combinatorial graph with nodes for event points and edges for the curve segments connecting them. We exploit the CAD of the sheared curve to compute the event graph; the event graph in turn bears enough information to construct the CAD for f. Note that the sheared images of event points of f are not necessarily critical points of $\mathrm{Sh}f$ (Figure 6.1), so they are not covered by the computed CAD of $\mathrm{Sh}f$ and must be detected additionally.

Let $P := \{p_1, \dots, p_r\}$ denote the set of event points of f. Our algorithm proceeds in four steps:

1. Find the sheared images $P^* := \{p_1^*, \dots, p_r^*\}$ of the event points of f.

2. Construct the sheared event graph $G^* := (P^*, E^*)$ with edges (p_i^*, p_j^*) for the segments of $\mathrm{Sh}f$ connecting p_i^* and p_j^*.

3. Compute the event graph $G = (P, E)$ by replacing sheared points p_i^* by their preimages p_i.

4. Using G, construct the stacks for the original curve f.

Step 1 is described in §6.1. Step 2 is straightforward, exploiting the CAD of $\mathrm{Sh}f$.

In Step 3, we compute the coordinates of p_i out of p_i^*. Note that y-coordinates of p_i and p_i^* are equal, so it is only about finding the correct x-coordinate for p_i. Since event points are critical, the x-coordinate of p_i must be a critical one, i.e., a root of the (already known) resultant $\mathrm{res}_y(f, \frac{\partial f}{\partial y})$. To find out which root it is, approximations of the x- and y-coordinate of p_i^* are computed and the inverse shear is applied using interval arithmetic, yielding an x-range of possible x-coordinates for p_i. The approximations are refined iteratively until the resulting x-range overlaps with exactly one isolating interval of the resultant roots.

Step 4 is described in more detail in §6.2. It employs another variant of the Bitstream Descartes method, again using extra information to terminate despite the presence of multiple roots. This extra information now comes from the event graph, and no symbolic computation is needed for this step.

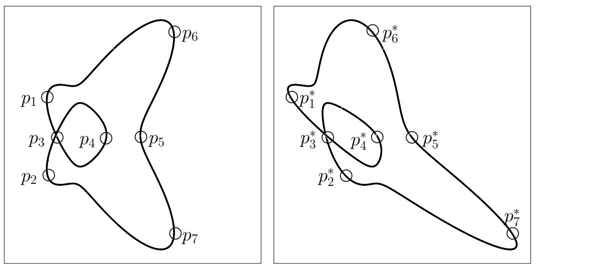

Figure 6.1: On the left: A (non-generic) curve of total degree four and its event points. On the right: Its sheared curve (with shear factor 2) and the sheared event points. Note that all p_i^* except p_3^* are non-critical points of the sheared curve.

6.1 Sheared event point detection

We search for the sheared event points of the curve f. We begin with their x-coordinates.

Lemma 6.2. *Let (α, β) be a sheared critical point, i.e., the image of a critical point of f under the shear. Then α is a root of $R_{\mathrm{ev}} := \mathrm{res}_y(\mathrm{Sh}f, \mathrm{Sh}\frac{\partial f}{\partial y})$.*

As event points are always critical, it is enough to search for sheared event points over each root of R_{ev}. We refine the CAD for $\mathrm{Sh}f$ by introducing new stacks at those roots α of R_{ev} at which no stack exists yet, by running the Bitstream Descartes method on the square free polynomials $(\mathrm{Sh}f)_\alpha$. Such newly created stacks subdivide some intermediate interval in two parts, and at least one new intermediate stack must be created as well.

Now we consider any point p^* over some root of R_{ev} and ask whether it is a sheared event point. If p^* does not have exactly two adjacent points in total in its two neighboring stacks, it obviously is an event point (e.g., the point p_3^* in Figure 6.1). However, what if it does have exactly two adjacent points? Let us call them q_1^* and q_2^*, and their preimages q_1 and q_2. Clearly, p, the preimage of p^*, is an event point if and only if q_1 and q_2 are both "on the same side" of p, hence one could just shear back p^*, q_1^* and q_2^* and compare their x-coordinates. We will derive a more efficient criterion which only depends on the q_i^*'s and does not shear back any point; cf. Figure 6.2.

Lemma 6.3. *The point p^* on $\mathrm{Sh}f$ is a sheared event point of f if and only if $\mathrm{sgn}((\mathrm{Sh}\frac{\partial f}{\partial y})(q_1^*)) \neq \mathrm{sgn}((\mathrm{Sh}\frac{\partial f}{\partial y})(q_2^*))$.*

Proof. Notice that $(\mathrm{Sh}\frac{\partial f}{\partial y})(q_i^*) = \frac{\partial f}{\partial y}(q_i)$. We let $q_i = (a_i, b_i)$ and observe that $\frac{\partial f}{\partial y}(q_i) = f'_{a_i}(b_i)$. Hence it suffices to show in the original system that p is an event point if and only if $\operatorname{sgn}(f'_{a_1}(b_1)) \neq \operatorname{sgn}(f'_{a_2}(b_2))$. The plane decomposes into the curve $f = 0$ and into *regions* (connected open subsets) that are positive ($f > 0$) or negative ($f < 0$). Consider the x-monotone segments σ_i of f that connect p to q_i, $i = 1, 2$.

If σ_1, σ_2 extend to different sides of p, then p is not an event point, and $\sigma := \sigma_1 \cup \sigma_2$ is x-monotone and separates two regions. W.l.o.g., let the region below σ be negative. A vertical upward ray at $x = a_i$ leaves this region at the simple root b_i of f_{a_i}, so the region above σ is positive, and $f'_{a_i}(b_i) > 0$ for both i.

If σ_1, σ_2 extend to the same side of p, then p is an event point, and w.l.o.g. there is a negative region above σ_1 and below σ_2. An upward vertical ray at $x = a_1$ enters this negative region at the simple root b_1 of f_{a_1}, hence $f'_{a_1}(b_1) < 0$. A similar ray at $x = a_2$ leaves this negative region at b_2, hence $f'_{a_2}(b_2) > 0$. □

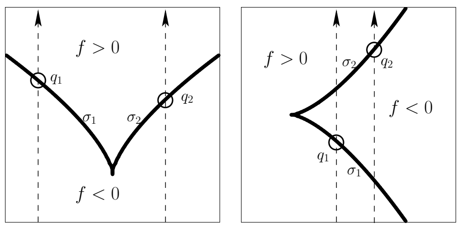

Figure 6.2: At the left: For $i = 1, 2$, the function f_{a_i} (dashed line) changes from the negative to the positive region, so $\frac{\partial f}{\partial y}(q_i) > 0$. On the right, it is $\frac{\partial f}{\partial y}(q_1) < 0$, $\frac{\partial f}{\partial y}(q_2) > 0$.

Consequently, to check whether the point p^* is a sheared event point, we only need one function evaluation at its two adjacent points. Since the sign is known to be non-zero, it can be determined numerically by approximating q_i^* sufficiently.

6.2 Stack construction

We explain the last step in our transformation algorithm next: We already have computed the graph G^*, containing the sheared event points and their connections, and for each sheared event point p_i^* we know the corresponding event point p_i. When each p_i^* is replaced by p_i in G^*, we obtain the event graph G that contains all event points of f and their connections (compare Figure 6.1). Moreover, two event points must be connected via an *x-monotone* segment of the curve since otherwise the segment would contain a further event point. This allows to count the number of points in each stack combinatorially.

Proposition 6.4. *Let $G = (V, E)$ be the event graph of f, and let α be a real root of $\operatorname{res}_y(f, \frac{\partial f}{\partial y})$. Let m_1 denote the number of event points with x-coordinate α, and m_2 the number of edges in the graph such that one endpoint has x-coordinate smaller α and one endpoint greater α. Then, the number of points over α is $m := m_1 + m_2$.*

As an example, consider the stack over p_4 in Figure 6.1: There is one event point, and the edges (p_1, p_6) and (p_2, p_7) cause additional points at α, so there are three roots in total. We also get the number of adjacent points at the left and right neighboring stack for each event point by the analogous counting argument.

The event graph does not indicate in which order event points and non-event points are arranged over α. We need to isolate the real roots of f_α as a final step. We already know m, the number of points over f_α, from the event graph. Also, we know m_1, the number of event points over α, and we can approximate their y-coordinates up to any precision, because we have the coordinates of their sheared images, and the y-coordinate does not change when shearing back.

We run the Bitstream Descartes algorithm for the non-square free f_α. During the subdivision process, those intervals that contain an event point are marked. If m_1 marked intervals are found, they all contain exactly one event point. We further subdivide until we find $m - m_1 = m_2$ unmarked intervals with an odd number of sign variations. Such an interval must contain at least one real root, and with the knowledge of m and m_1, we can stop and report the isolating intervals.

It remains to show that eventually m_2 unmarked intervals with *odd* sign variation are found. This follows from the next lemma.

Lemma 6.5. *Let (α, β) be a non-event point. Then, β is a root of f_α with odd multiplicity.*

Proof. By using the same argument as in the proof of Lemma 6.3, the function f_α changes its sign at the root β (see Figure 6.2(left)). □

6.3 Unbounded arcs

We now discuss the case that f is unbounded, which we have omitted so far. An unbounded arc can either be unbounded in x-direction, we say then it goes to $x = -\infty$ or $x = +\infty$, or it is bounded in x-direction, but unbounded in y-direction. In that case, the arc converges to the vertical asymptote $x = \alpha$ for some critical value α, and we call the arc asymptotic arc for α. Asymptotic arcs either go to $y = +\infty$ or $y = -\infty$; we say the arc goes up or down.

For brevity, we only sketch the treatment: Unbounded arcs of f and $\mathrm{Sh}f$ are in one-to-one correspondence, and the latter curve only has unbounded arcs to $x = \pm\infty$ by genericity condition (G1). To deduce the type of an unbounded arc of f, we choose a point on each unbounded arc of $\mathrm{Sh}f$ which is "far" enough out towards $x = \pm\infty$, and we shear it back to the original system. The position of this sheared point determines the type of the unbounded arc. See [20, §5.4.3] for details. The information about the unbounded arcs is stored in the event graph, introducing nodes with symbolic coordinates for points at infinity.

7. IMPLEMENTATION, EXPERIMENTS

We have implemented our method as the C++ library `AlciX` as part of the EXACUS project[1] [5]. `AlciX` consists of about 8 000 lines of code, not counting the supporting code from other EXACUS libraries. We implemented Ducos' algorithm [10] to compute Sturm-Habicht sequences. Gcds of univariate integer polynomials are computed with Shoup's NTL[2]. For exact integer arithmetic, we use the GMP library[3]. We report on experiments performed on a machine

[1]`http://www.mpi-inf.mpg.de/projects/EXACUS/`
[2]`http://www.shoup.net/ntl/`
[3]`http://www.swox.com/gmp/`

with a Pentium 4 CPU clocked at 2.80 GHz and 1 GB of RAM.

7.1 Comparison with a topology algorithm

We have compared `AlciX` with an algorithm that computes the topology of algebraic curves: the algorithm `top` from Gonzalez-Vega and Necula [15], implemented in MAPLE. As input, we used the 16 curves from [15], subsequently denoted by gn_1 to gn_{16}. We remark that `top` can be supplied with an initial precision for floating point calculations. To achieve best performance of `top`, we selected sufficient precisions based on [15, Tbl. 1]. We ran the program on MAPLE Version 10.

Table 1: Comparison of `AlciX` and `top`. All timings are given in seconds, the numerical precision of `top` is given in parentheses.

	AlciX	top		AlciX	top
gn_1	0.220	0.987 (20)	gn_9	0.194	0.333 (15)
gn_2	0.012	0.097 (15)	gn_{10}	0.177	0.150 (10)
gn_3	0.011	0.102 (15)	gn_{11}	0.086	0.591 (15)
gn_4	0.072	0.079 (10)	gn_{12}	0.326	19.207 (40)
gn_5	0.043	0.068 (10)	gn_{13}	0.008	0.069 (10)
gn_6	0.217	0.386 (20)	gn_{14}	0.338	0.811 (30)
gn_7	0.036	0.083 (10)	gn_{15}	0.006	0.024 (10)
gn_8	0.028	0.181 (20)	gn_{16}	0.104	0.125 (10)

We see in Table 1 that `AlciX` is considerably faster than `top` in almost all cases. In particular, notice the improvement by factor 59 for the curve gn_{12}. It must be taken into account that `AlciX` is implemented in a more performance-optimized programming language, but the fact that `AlciX` computes more than the topology might compensate this partially.

7.2 Comparison with a CAD algorithm

In this section, we compare the running time of `AlciX` with Brown's `cad2d`, an optimized version of `QEPCAD-B`[4] (Version 1.46) for computing CADs in the plane. Its advantage over the more general `QEPCAD-B` is that it uses floating point methods in the lifting step to simplify calculations in favorable situations. Brown describes such optimizations in [6]. `cad2d` is able to produce CADs for an arbitrary number of curves, but we restrict to one curve for the comparison with our method.

By default, `cad2d` does not compute the adjacencies of the computed CAD. This computation however can be forced by a subsequent call of the `closure2d` command that computes adjacencies as a first step.[5]

We tried `cad2d` also for the polynomials $gn_1 - gn_{16}$ from the previous section. The running times of `cad2d` and `AlciX` were about the same in most cases, but sometimes `cad2d` was considerably faster (factor 6 for gn_{14}), or slower ($>$ 3 seconds for gn_8) compared to `AlciX`. We perform more systematic tests, using a similar setup as in [6].

We begin by comparing the running times for polynomials with random coefficients and 50 percent term density. For each degree, we created five test polynomials. See Table 2 for running times. We called `cad2d` with option `+N10000000` since it runs out of memory with the default settings for some instances.

Table 2: Timings for random curves

		10 bit coefficients		50 bit coefficients	
deg		AlciX	cad2d	AlciX	cad2d
9	a	0.162	0.134	0.285	0.408
	b	*0.577	0.099	0.216	0.651
	c	0.078	0.103	*1.005	0.274
	d	0.083	0.112	0.426	0.659
	e	0.126	0.171	0.232	0.309
12	a	*3.577	0.429	1.603	3.036
	b	0.272	0.670	1.736	1.610
	c	0.609	0.450	*7.337	1.331
	d	*2.351	0.486	*7.494	[1]Fl. point
	e	0.779	0.410	*7.722	Fl. point
15	a	2.653	1.195	7.894	4.095
	b	1.840	2.064	9.149	6.430
	c	1.775	1.410	8.580	5.665
	d	*15.554	2.489	*40.090	4.200
	e	1.609	1.905	7.649	3.431

*A shear has been applied in `AlciX`.
[1]`cad2d` reported a floating point exception.

Two weaknesses of `AlciX` become visible here: First, a coordinate change results in additional calculations (e.g. one needs three resultants instead of one), and moreover causes longer coefficients for the transformed polynomial. Second, `AlciX` always computes the full Sturm-Habicht sequence (accounting for more than two thirds of the time in the 50 bits / degree 15 examples). In contrast `cad2d` only needs the resultant and computes it with a fast modular method.

However, the simplifications of `cad2d` fail for the more interesting case of curves with singular points, and `AlciX` also makes use of numerical simplifications in those situations. We construct plane curves as resultants from randomly generated trivariate polynomials p, q with 50 percent term density and 8 bit coefficients. Usually, the resultant is a dense polynomial with degree $\deg_z(p) \cdot \deg_z(q)$ and the curve it defines contains singularities [22]. The running times in Table 3 show that `AlciX` computes the cad of such singular curves much faster than `cad2d` in general.

8. CONCLUSION

We have presented an algorithm for the geometric analysis of algebraic plane curves. Its design was guided by the goal of reducing symbolic computations, for the sake of efficiency, without compromising exactness of the result. The only symbolic operations performed during the execution of our algorithm are: (1) computing principal Sturm-Habicht coefficients and resultants of integer polynomials, (2) isolating multiple real roots of resultants, and (3) gcd computation for the zero test (during sign determination) of $\mathrm{stha}_i(f)(x)$, evaluated at multiple real roots of the resultant. The other operations with algebraic numbers are done numerically with sufficient precision. In particular, we use the newly introduced m-k-Descartes method for the lifting phase and remove the need for exact arithmetic as in the Bitstream Descartes method [12]. This works for all inputs, exact arithmetic as a fall-back is not necessary. The m-k-

[4]`http://www.cs.usna.edu/~qepcad/B/QEPCAD.html`
[5]We thank Christopher Brown for this advice.

Table 3: Running times for resultants of random trivariate polynomials.

degrees		AlciX	cad2d
(3,3) $\rightsquigarrow$ 9	a	2.358	0.235
	b	0.183	1.241
	c	0.209	1.874
	d	0.190	0.263
	e	0.092	0.243
(3,4) $\rightsquigarrow$ 12	a	1.228	49.960
	b	1.510	66.938
	c	*6.462	78.434
	d	1.728	90.945
	e	0.798	7.780
(4,4) $\rightsquigarrow$ 16	a	12.042	[2]Prime list
	b	12.871	Prime list
	c	6.972	795.376
	d	12.296	Fl. point
	e	13.121	Prime list

[2]The prime list of `cad2d` has been exhausted during the computation.

Descartes method combines an initial exact count of distinct real roots with the less precise but efficient counting done by Descartes' rule during the interval subdivision. With the m-k-Descartes method, we detect problematic non-generic situations along the way, so no separate expensive genericity check is necessary. Since roots are counted with multiplicity, the method also identifies a unique interval (the candidate) in which a multiple root may be contained.

We also circumvent certain symbolic computations in the transformation phase by searching only for sheared event points, which are more efficiently detectable than sheared critical points in general.

Our method does not need any global precision control: the Bitstream Descartes method chooses an appropriate precision internally; other non-symbolic computations with algebraic numbers are performed in interval arithmetic, refining the initial intervals until the result is sufficiently closely approximated.

The experiments (Section 7) show that our cutback of symbolic operations is successful. `AlciX` outperforms `cad2d` for curves with singular points, presumably because `AlciX` uses numerical lifting consistently whereas the optimizations of `cad2d` apply only in simple cases. `AlciX` also outperforms `top` for the large majority of tested examples.

Our approach can be extended to the analysis of two curves, leading to an optimized CAD algorithm in the plane for an arbitrary number of polynomials.

Acknowledgements. The authors thank Kurt Mehlhorn, Susanne Schmitt and Raimund Seidel for useful discussions, and Eric Berberich and Michael Hemmer for their help on the implementation of `AlciX`. We thank Laureano Gonzalez-Vega and Ioana Necula for making `top` available for our experiments. We also thank the anonymous referees for valuable comments.

9. REFERENCES

[1] J. Abbott: "Quadratic Interval Refinement for Real Roots". URL `http://www.dima.unige.it/~abbott/`. Poster presented at the 2006 Int. Symp. on Symb. and Alg. Comp. (ISSAC 2006).

[2] D. Arnon, G. Collins, S. McCallum: "Cylindrical Algebraic Decomposition I: the Basic Algorithm". *SIAM J. Comp.* **13** (1984) 865–877.

[3] D. Arnon, G. Collins, S. McCallum: "Cylindrical Algebraic Decomposition II: an Adjacency Algorithm for the Plane". *SIAM J. Comp.* **13** (1984) 878–889.

[4] S. Basu, R. Pollack, M.-F. Roy: *Algorithms in Real Algebraic Geometry*. Springer, 2nd edn., 2006.

[5] 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". In: *Proc. of the 13th Ann. European Symp. on Alg. (ESA 2005), LNCS*, vol. 3669. Springer, 2005 155–166.

[6] C. W. Brown: "Constructing Cylindrical Algebraic Decompositions of the Plane Quickly", 2002. URL `http://www.cs.usna.edu/~wcbrown/`. Unpublished.

[7] G. Collins: "Quantifier Elimination For Real Closed Fields By Cylindrical Algebraic Decomposition". In: *Proc. 2nd GI Conf. on Automata Theory and Formal Languages, LNCS*, vol. 33. Springer, 1975 134–183. Reprinted with corrections in: B. F. Caviness, J. R. Johnson (eds.), *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pp. 85–121, Springer, 1998.

[8] G. Collins, A. Akritas: "Polynomial Real Root Isolation Using Descartes' Rule of Signs". In: R. Jenks (ed.) *Proc. of the third ACM symp. on Symb. and Alg. Comp.* ACM, 1976 272–275.

[9] G. Collins, J. Johnson, W. Krandick: "Interval Arithmetic in Cylindrical Algebraic Decomposition". *J. Symb. Comp.* **34** (2002) 143–155.

[10] L. Ducos: "Optimizations of the Subresultant Algorithm". *J. Pure Appl. Alg.* **145** (2000) 149–163.

[11] A. Eigenwillig: "On Multiple Roots in Descartes' Rule and Their Distance to Roots of Higher Derivatives"'. *J. Comp. Appl. Math.* **200** (2007) 226–230.

[12] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, N. Wolpert: "A Descartes Algorithm for Polynomials with Bit-Stream Coefficients". In: *8th Int. Workshop on Comp. Alg. in Scient. Comp. (CASC 2005), LNCS*, vol. 3718, 2005 138–149.

[13] A. Eigenwillig, V. Sharma, C. Yap: "Almost Tight Recursion Tree Bounds for the Descartes Method". In: *Proc. of the 2006 Int. Symp. on Symb. and Alg. Comp. (ISSAC 2006)*. ACM, 2006 71–78.

[14] L. Gonzalez-Vega, M. El Kahoui: "An Improved Upper Complexity Bound for the Topology Computation of a Real Algebraic Plane Curve". *J. Compl.* **12** (1996) 527–544.

[15] L. Gonzalez-Vega, I. Necula: "Efficient Topology Determination of Implicitly Defined Algebraic Plane Curves". *Comp. Aided Geom. Design* **19** (2002) 719–743.

[16] L. Gonzalez-Vega, T. Recio, H. Lombardi, M.-F. Roy: "Sturm-Habicht Sequences, Determinants and Real Roots of Univariate Polynomials". In: B. Caviness, J. Johnson (eds.) *Quantifier Elimination and Cylindrical Algebraic Decomposition*, 300–316. Springer, 1998.

[17] H. Hong: "An Efficient Method for Analyzing the Topology of Plane Real Algebraic Curves". *Math. and Comp. Sim.* **42** (1996) 571–582.

[18] J. R. Johnson: "Algorithms for polynomial real root isolation". In: B. F. Caviness, J. R. Johnson (eds.) *Quantifier Elimination and Cylindrical Algebraic Decomposition*, 269–299. Springer, 1998.

[19] J. R. Johnson, W. Krandick, K. Lynch, D. G. Richardson, A. D. Ruslanov: "High-Performance Implementations of the Descartes Method". In: *Proc. of the 2006 Int. Symp. on Symb. and Alg. Comp. (ISSAC 2006)*. ACM, 2006 154–161.

[20] M. Kerber: *Analysis of Real Algebraic Plane Curves*. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

[21] W. Krandick, K. Mehlhorn: "New Bounds for the Descartes Method". *J. Symb. Comp.* **41** (2006) 49–66.

[22] S. McCallum: "Factors of Iterated Resultants and Discriminants". *J. Symb. Comp.* **27** (1999) 367–385.

[23] F. Rouillier, P. Zimmermann: "Efficient isolation of [a] polynomial's real roots". *J. Comp. and Appl. Math.* **162** (2004) 33–50.

[24] R. Seidel, N. Wolpert: "On the Exact Computation of the Topology of Real Algebraic Curves". In: *Proc. of the 21st Ann. ACM Symp. on Comp. Geom. (SCG 2005)*. ACM, 2005 107–115.

[25] A. Strzebonski: "Cylindrical Algebraic Decomposition using validated numerics". *J. Symb. Comp.* **41** (2006) 1021–1038.