Confirmation Report

Corin Lee

October 25, 2022

1 Introduction

My thesis topic concerns cylindrical algebraic decomposition (CAD), which is a computational technique in real algebraic or semi-algebraic geometry. In this report I will give a short account of CAD and of the algorithms used for it. I will concentrate on the relatively new RC-CAD, which is implemented but largely unexplored theoretically. I will explain roughly how it works and what the immediate problems are, and outline the work that I intend to undertake. The idea is to explore RC-CAD in more detail and to come to some understanding of the theoretical complexity and is and what the worst cases are likely to be. Similar analyses have been carried out for the much better studied PL-CAD algorithms. Beyond that lie further questions such as whether modified versions of the algorithm might perform better in certain cases, and how they compare with PL-CAD.

A *cylindrical algebraic decomposition* (abbreviated CAD) is a method in real semi-algebraic geometry to decompose $\mathbb{R}^n$ into a finite number of connected semi-algebraic subsets known as *cells*, each homeomorphic to $\mathbb{R}^k$.

First arising in Collins' paper as a sub-algorithm in his work on an effective method for quantifier elimination in real closed fields [Col75], CAD has applications in algebraic simplification technology [BD02] and robot motion planning, one example of which is the Piano Movers Problem [Dav86, WDEB13], defined in [SS83a] as:

"Given a body B, and a region bounded by a collection of "walls", either find a continuous motion connecting two given positions and orientations of B during which B avoids collision with the walls, or else establish that no such motion exists."

In [SS83a], the two-dimensional case is considered. Later, in [SS83b], this problem is looked at in higher dimensions and a general method is detailed, solved using the CAD algorithm. However, as shown in [Dav86], trying to actually do this even for a simple example (the case of a ladder of length 3 moving through a right-angled corridor of with 1) can be computationally expensive and impractical - projecting this results in 250 distinct univariate polynomials of degree up to 26, and producing the CAD was not possible.

Wilson [WDEB13] was able to produce a CAD for this with a new formulation of the problem, but this still took around 5 hours using QEPCAD and produced a

CAD with over 285,000 cells.

In fact, it is known that the worst-case complexity is double-exponential in the number of variables [DH88]. Nevertheless, the algorithms are practicable in important cases, and small efficiency gains become important. For example, if the region to be decomposed lies in a subvariety (equational constraint), then this can be exploited but technical problems arise. In the recent thesis of Nair [Nai21], the author writes:

"McCallum attempted to tackle [this] first by switching from sign invariant CADs to order invariant CADs and then by exploiting equational constraints in the input formulae."

The author then proceeds to tackle some of the difficulties that arise in that context.

There are even competing definitions of CAD and types of CAD, mentioned below.

1.1 Definitions

Let $\mathbb{R}[\mathbf{x}]$ be the polynomial ring over $\mathbb{R}$ with ordered variables $\mathbf{x} = x_1 < \cdots < x_n$.

The following definitions are taken from the works of Arnon et. al [ACM84, ACM98] and Jirstrand [Jir95].

Definition 1.1 (region, cylinder, section, sector)**.** For n-dimensional real space $\mathbb{R}^n$,

- A nonempty connected subset R of $\mathbb{R}^n$ is known as a *region*.
- For a region R, the *cylinder* over R, written $\mathcal{Z}(R)$, is the set $R \times \mathbb{R} = \{(\alpha, x) \mid \alpha \in R, x \in \mathbb{R}\}$.
- For f a continuous, real valued function of R, an *f-section* of $\mathcal{Z}(R)$ is the set $\{\alpha, f(\alpha) \mid \alpha \in R\}$
- For f_1, f_2 continuous, real valued functions of R, an *(f_1, f_2)-sector* of $\mathcal{Z}(R)$ is the set $\{(\alpha, \beta) \mid \alpha \in R, f_1(\alpha) < \beta < f_2(\alpha)\}$. The functions $f_1 = -\infty$ and $f_2 = +\infty$ are allowed.

Definition 1.2 (Decomposition, stack)**.** For any subset X of $\mathbb{R}^n$, a *decomposition* of X is a finite collection of disjoint regions whose union is X.

Continuous, real-valued functions $f_1 < f_2 < \cdots < f_k$, $k \geq 0$, defined on R, naturally determine a decomposition of $\mathcal{Z}(R)$ consisting of the following regions:

(1) the (f_i, f_{i+1})-sectors of $\mathcal{Z}(R)$ for $0 \leq i \leq k$, where $f_0 = -\infty$ and $f_{k+1} = +\infty$, and

(2) the f_i-sections of $\mathcal{Z}(R)$ for $1 \leq i \leq k$.

We call such a decomposition a *stack* over R (determined by $f_1, \ldots, f_k$).

Definition 1.3 (Cylindrical decomposition of $\mathbb{R}^n$)**.** A decomposition $\mathcal{D}$ of $\mathbb{R}^n$ is *cylindrical* if either

(1) $n = 1$ and $\mathcal{D}$ is a stack over $\mathbb{R}^0$, or

(2) $n > 1$, and there is a unique cylindrical decomposition $\mathcal{D}'$ of $\mathbb{R}^{n-1}$ such that for each region R of $\mathcal{D}'$, some subset of $\mathcal{D}$ is a stack over R.

As $\mathcal{D}'$ is unique, any cylindrical decomposition $\mathcal{D}$ of $\mathbb{R}^n$ will have unique *induced* cylindrical decompositions of $\mathbb{R}^j$ for $j = n-1, n-2, \ldots, 1$. Conversely, given a CAD $\mathcal{D}'$ of $\mathbb{R}^j, j < n$, a CAD $\mathcal{D}$ of $\mathbb{R}^n$ is an *extension* of $\mathcal{D}'$ if $\mathcal{D}$ induces $\mathcal{D}'$.

Alternatively, a decomposition is *cylindrical* if for all $1 \leq j < n$, the projections on the first j variables of any two cells are either equal or disjoint.

Definition 1.4 (Semi-algebraic set)**.** A subset of $\mathbb{R}^n$ is *semi-algebraic* if it can be constructed by finitely many applications of the union, intersection, and complementation operations on sets of the form

$$\{x \in \mathbb{R}^n \mid f(x) \geq 0\},$$

where $f \in \mathbb{R}[\mathbf{x}]$.

Definition 1.5 (Cylindrical Algebraic Decomposition)**.** A decomposition is *algebraic* if each of its regions is a semi-algebraic set. *A cylindrical algebraic decomposition*, or *CAD*, of $\mathbb{R}^n$ is a decomposition which is both cylindrical and semi-algebraic.

The components of a CAD are called *cells*, and for $0 \leq j \leq n$, a *j-cell* in $\mathbb{R}^n$ is a subset of $\mathbb{R}^n$ which is homeomorphic to $\mathbb{R}^j$.

We often want the decomposition to respect some collection of polynomials:

Definition 1.6 ($\mathcal{F}$-invariant)**.**
Let $\mathcal{F} = \{f_i \in \mathbf{k}[x_1, \ldots, x_n], 1 \leq i \leq r\}$ be a set of polynomials in $\mathbf{k}[x_1, \ldots, x_n]$ for $\mathbf{k} = \mathbb{C}$ or $\mathbb{R}$ and $X \subseteq \mathbf{k}^n$. We say X is *$\mathcal{F}$-invariant* (and $\mathcal{F}$ is *invariant* on X) if each $f_i(x)$ has constant sign for every $x \in X$, that is,

$$\forall x \in X : f_i(x) \ \diamond \ 0,$$

where for $\mathbf{k} = \mathbb{R}, \diamond \in \{>, =, <\}$, and for $\mathbf{k} = \mathbb{C}, \diamond \in \{=, \neq\}$. In other words, f_i is either identically zero or never zero; a polynomial $f_i(x)$ with constant sign in this sense is called *sign-invariant*.

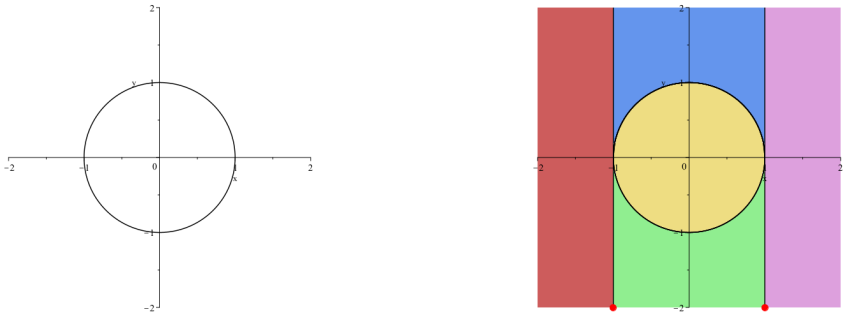

Figure 1: The graph of $x^2 + y^2 - 1$ and the $\{x^2 + y^2 - 1\}$-invariant CAD of $\mathbb{R}^2$

2 Approaches and Implementations

There have been several implementations of Collins' original projection and lifting CAD algorithm (hereby referred to as PL-CAD) in various software, including [Bro, Res, Red, YA07, Eng, Ton21], along with various improvements. Another algorithm, based on triangular sets and regular chains (hereby referred to as RC-CAD), has been implemented in Maple as part of the `RegularChains` package [CMM]. Despite this, little appears to have been done in terms of complexity analysis of this algorithm.

2.1 PL-CAD

Projection and lifting algorithms work by defining a *projection operator* which takes a set of polynomials and produces another set of polynomials in one fewer variables. This projection typically involves coefficients, discriminants, resultants and subresultants. This is applied recursively until one reaches a cylindrical decomposition of $\mathbb{R}^1$, which is a decomposition into intervals. An appropriate lifting algorithm is used to build the full CAD and return to the "top" dimension. See, for example [Nai21] and the sources therein.

The algorithm takes a set $\mathcal{F}$ of polynomials in $\mathbb{R}[x_1, \ldots, x_n]$ and outputs an $\mathcal{F}$-invariant CAD of $\mathbb{R}^n$.

The PL-CAD algorithm can be split into three phases:

- A projection phase, which uses the *projection operator* Proj to take $\mathcal{F} = \mathcal{F}_n$ from a subset of $\mathbb{R}[x_1 < \cdots < x_n]$ to a subset $\mathcal{F}_{n-1}$ of $\mathbb{R}[x_1 < \cdots < x_{n-1}]$ recursively down to $\mathbb{R}^1$. The zero sets of the polynomials produced by each step are the projections of "significant points" of the previous set of polynomials, such as self-crossings, vertical tangent points, isolated points etc.
- A base phase consisting of real root isolation on $\mathbb{R}^1$ on this output, where these roots and the open intervals between form an $\mathcal{F}_1$-invariant CAD of $\mathbb{R}^1$.
- A lifting phase which, for each cell C of the $\mathcal{F}_{k-1}$-invariant CAD in $\mathbb{R}^{k-1}$, involves constructing a sample point s and isolating the real roots of the polynomials of $\mathcal{F}_k$ at s. The sectors and sections of these polynomials form a stack, and these stacks make the cells of the $\mathcal{F}_k$-invariant CAD of $\mathbb{R}^k$ above C.

It is enough to determine the signs of $\mathcal{F}$ in these sample points as each cell is $\mathcal{F}$-invariant by construction.

Different CAD algorithms of this type may use different projections and different lifting algorithms, depending on their characteristics. Examples are Collins' original algorithm [Col75] and variants due to McCallum, Lazard, Brown and Nair [McC84, Laz94, Bro01, Nai21].

Typically these variants aim at producing better performance under certain circumstances, see [Nai21] for more details. Similar questions could arise in the future for RC-CAD (see [CMM12]).

2.2 RC-CAD

There has been much work by Chen and Moreno Maza on an incremental algorithm for computing CADs using triangular systems and regular chains (RC-CAD), by creating a complex cylindrical tree and refining it into a real cylindrical tree.

Much has been learnt about PL-CAD over the last forty years, with many enhancements made along the way. Although there has been a lot of work implementing RC-CAD into *Maple* via the `RegularChains` package, little analysis has been done. Therefore we are interested in trying to understand how the algorithm works in sufficient detail to give estimates and boundaries for its complexity, and in learning when and where the algorithm is more, or less, efficient.

In [CMMXY09] the authors presented an alternative way to compute CADs using triangular sets, by constructing a cylindrical decomposition of the complex space, or CCD, from which one can easily produce a CAD. This had the advantage that other PL-CAD methods did not have, which is that it did not have problems with curtains [Nai21], that is, regions where a polynomial nullifies over a set.

Despite these advantages, the authors later mention in [CMM12], it involved "many black boxes, which hide many unneccesary or redundant computations". This gave a much higher computation time in tests than PL-CAD, despite usually computing fewer cells.

In [CMM12], the authors instead computed CCDs in an incremental way to avoid redundant computations, which we will now describe in more detail.

The CCD construction in RC-CAD can be seen as an enhanced projection phase of PL-CAD [CMM12], with the benefit that its "case discussion" scheme avoids unnecessary computations that the projection operator performs on unrelated branches, and avoids curtains.

The incremental algorithm of [CMM12] involves refining the branches of a tree via GCD computation. The CCD algorithm produces a decomposition into triangular sets, say $\mathcal{D}$ such that the zero sets of the output regular chains are disjoint. Such a tree is encoded by a tree data structure, and the decomposition computed is both disjoint and cylindrically arranged.

The complexity of this algorithm can also not be better than doubly exponential in the number of variables [BD02], but the benchmarking of [CMM12] shows PL-CAD outperforming QEPCAD and `Mathematica` for several well-known examples. Despite this, no theoretical complexity analysis seems to have been done, despite both these results and the availability of the algorithm in the `RegularChains` Maple package.

3 Description of RC-CAD

In [CMMXY09, CMM12] the following are defined over a field $\mathbf{k}$ of characteristic zero and $\mathbf{K}$ its algebraic closure, but for simplicity we will restrict ourselves to $\mathbb{R}$ and $\mathbb{C}$ respectively.

Let $p \in \mathbb{R}[\mathbf{x}]$ be a non-constant polynomial and $x \in \mathbf{x}$ be a variable.

1. We denote by $\deg(p, x)$ and $\mathrm{lc}(p, x)$ the *degree* and the *leading coefficient* of p w.r.t. x.
2. The greatest variable appearing in p is called the *main variable*, denoted by $\mathrm{mvar}(p)$.
3. The *separant* $\mathrm{sep}(p)$ of p is $\partial p / \partial \,\mathrm{mvar}(p)$.
4. The leading coefficient, the degree, and the reductum (p minus its leading term) of p regarded as a univariate polynomial in $\mathrm{mvar}(p)$ are called the *initial*, the *main degree*, the *tail* of p; they are denoted by $\mathrm{init}(p), \mathrm{mdeg}(p), \mathrm{tail}(p)$ respectively.
5. The integer k such that $x_k = \mathrm{mvar}(p)$ is called the *level* of the polynomial p.

3.1 Triangular Sets and Regular Chains

Definition 3.1 (Triangular set)**.** Let $\mathcal{T} \subset \mathbb{R}[\mathbf{x}]$ be a *triangular set*, that is, a set of nonconstant polynomials with pairwise distinct main variables, that is, for all $t, t' \in \mathcal{T}, \mathrm{mvar}(t) \neq \mathrm{mvar}(t')$.

We denote by $\mathrm{mvar}(\mathcal{T})$ the set of the main variables of the polynomials in $\mathcal{T}$.

A variable in $\mathbf{x}$ is called *algebraic* w.r.t. $\mathcal{T}$ if it belongs to $\mathrm{mvar}(\mathcal{T})$, otherwise it is said *free* w.r.t. $\mathcal{T}$.

For $v \in \mathbf{x}$, we denote by $\mathcal{T}_{<v}$ the set of the polynomials $t \in \mathcal{T}$ such that $\mathrm{mvar}(t) < v$ holds.

Let $h \in \mathbb{R}[\mathbf{x}]$. The *iterated resultant* of h w.r.t. $\mathcal{T}$, denoted by $\mathrm{ires}(h, \mathcal{T})$, is defined as follows:

(1) if $h \in \mathbb{R}$ or all variables in h are free w.r.t. $\mathcal{T}$, then $\mathrm{ires}(h, T) = h$;

(2) Otherwise, if v is the largest variable of h which is algebraic w.r.t. $\mathcal{T}$, then $\mathrm{ires}(h, \mathcal{T}) = \mathrm{ires}(r, \mathcal{T}_{<v})$ where r is the resultant w.r.t. v of h and the polynomial in $\mathcal{T}$ whose main variable is v.

Definition 3.2 (Regular chain, regular system)**.** Let $h_{\mathcal{T}}$ be the product of the initials of the polynomials in $\mathcal{T}$. A triangular set T is called a *regular chain* if either $\mathcal{T} = \emptyset$ or $\mathrm{ires}(h_{\mathcal{T}}, \mathcal{T}) \neq 0$. The pair $[\mathcal{T}, h]$ is called a *regular system* if $\mathcal{T}$ is a regular chain, and $\mathrm{ires}(h, \mathcal{T}) \neq 0$.

Definition 3.3 (Squarefree)**.** Denote by $\mathrm{sep}(\mathcal{T})$ the product of all $\mathrm{sep}(p)$, for $p \in \mathcal{T}$. Then $\mathcal{T}$ is said to be *squarefree if* $\mathrm{ires}(\mathrm{sep}(\mathcal{T}), \mathcal{T}) \neq 0$. A regular system $rs = [T, h]$ is said to be *squarefree* if $\mathcal{T}$ is squarefree.

3.2 Complex Cylindrical Trees

Definition 3.4 (Separation)**.** Let C be a subset of $\mathbb{C}^{n-1}$ and $P \subset \mathbb{R}[x_1, \ldots, x_{n-1}, x_n]$ be a finite set of level n polynomials. We say that P *separates above* C if for each

$\alpha \in C$:

- for each $p \in P$, the polynomial $\mathrm{init}(p)$ does not vanish at α,
- the polynomials $p(\alpha, x_n) \in \mathbb{C}[x_n]$, for all $p \in P$, are squarefree and coprime.

Note that this definition allows C to be a semi-algebraic set, see [CMM12].

Effectively, this says that the main variable of $p \in P$ is the same whether you look everywhere or only above C, so C is in "general position" relative to P.

Definition 3.5 (Cylindrical decomposition of $\mathbb{C}^n$ and associated tree)**.** By induction on n, we define the notion of a *cylindrical decomposition of* $\mathbb{C}^n$ together with that of the *tree associated with a cylindrical decomposition of* $\mathbb{C}^n$.

For $n = 1$, a cylindrical decomposition of $\mathbb{C}$ is a finite collection of sets $\mathcal{D} = \{D_1, \ldots, D_{r+1}\}$, where either $r = 0$ and $D_1 = \mathbb{C}$, or $r > 0$ and there exists r non-constant coprime squarefree polynomials $p_1, \ldots, p_r$ of $\mathbb{R}[x_1]$ such that for $1 \leq i \leq r$ we have

$$D_i = \{x_1 \in \mathbb{C} \mid p_i(x_1) = 0\}, \qquad \text{and} \qquad D_{r+1} = \{x_1 \in \mathbb{C} \mid p_1(x_1) \cdots p_r(x_1) \neq 0\}.$$

Note that the D_i, for all $1 \leq i \leq r+1$, form a partition of $\mathbb{C}$. The tree associated with $\mathcal{D}$ is a rooted tree whose nodes, other than the root, are $D_1, \ldots, D_r, D_{r+1}$ which all are leaves and children of the root.

Now let $n > 1$, and let $\mathcal{D}' = \{D_1, \ldots, D_s\}$ be any cylindrical decomposition of $\mathbb{C}^{n-1}$. For each D_i , let r_i be a non-negative integer and let $\{p_{i,1}, \ldots, p_{i,r_i}\}$ be a set of polynomials which separates above D_i. If $r_i = 0$, set $D_{i,1} = D_i \times \mathbb{C}$. If $r_i > 0$, set

$$D_{i,j} = \{(\alpha, x_n) \in \mathbb{C}^n \mid \alpha \in D_i \text{ and } p_{i,j}(\alpha, x_n) = 0\},$$

for $1 \leq j \leq r_i$ and set

$$D_{i,r_i+1} = \{(\alpha, x_n) \in \mathbb{C}^n \mid \alpha \in D_i \text{ and } \prod_{j=1}^{r_i} p_{i,j}(\alpha, x_n) \neq 0\}.$$

The collection $\mathcal{D} = \{D_{i,j} \mid 1 \leq i \leq s, \quad 1 \leq j \leq r_i + 1\}$ is called a *cylindrical decomposition* of $\mathbb{C}^n$. The sets $D_{i,j}$ are called the *cells* of $\mathcal{D}$.

If T' is the tree associated with $\mathcal{D}'$ then the tree T associated with $\mathcal{D}$ is defined as follows. For each $1 \leq i \leq s$, the set D_i is a leaf in T' which has all $D_{i,j}$ for children in T; thus the $D_{i,j}$ are the leaves of T.

Note that each node N of T is either associated with no constraints, or associated with a polynomial constraint, which itself is either an equation or an inequation. Note also that, if the level of the polynomial defining the constraint at N is ℓ, then ℓ is the length of a path from N to the root.

Moreover, the polynomial constraints along a path from the root to a leaf form a polynomial system called a *cylindrical system of* $\mathbb{R}[x_1, \ldots, x_{n-1}]$ *induced by* T. Let S be such a cylindrical system. We denote by $Z(S)$ the *zero set* of S. Therefore, each cell of $\mathcal{D}$ is the zero set of a cylindrical system induced by T.

Let $\hat{T}$ be a sub-tree of T such that the root of $\hat{T}$ is that of T. Then, we call $\hat{T}$ a *cylindrical tree of* $\mathbb{R}[x_1, \ldots, x_{n-1}]$ *induced by* T. This cylindrical tree $\hat{T}$ is said to be *partial* if it admits a non-leaf node N such that the zero set of the constraint of N is not equal to the union of the zero sets of the constraints of the children of N. If $\hat{T}$ is not partial, then it is called *complete*.

Let $\mathcal{F} = \{f_1, \ldots, f_s\}$ be a finite set of polynomials of $\mathbb{R}[\mathbf{x}]$. A cylindrical decomposition $\mathcal{D}$ of $\mathbb{C}^n$ is called *$\mathcal{F}$-invariant* if for any given cell D of $\mathcal{D}$ and any given polynomial $f \in \mathcal{F}$, either f vanishes at all points of D or f vanishes at no points of D, and f is *sign invariant* if it is either identically zero or invertible. Note that this is consistent with Definition 1.6 from earlier in this paper.

The definition of $Z(S)$ can be extended by replacing S with a subtree $\hat{T}$ as above:

$Z(\hat{T}) = \{w \in \mathbb{C}^n \mid f(w) = 0 \ \forall f \text{ labelling edges of } \hat{T}\}$

If $p \in \mathbb{R}[\mathbf{x}]$, we denote by V_p the variety of p, and we say

1. p is *invertible modulo* $\hat{T}$ if $V_p \cap Z(\hat{T}) = \emptyset$.
2. p is *zero modulo* $\hat{T}$ if $V_p \cap Z(\hat{T}) = Z(\hat{T})$.
3. We say p is *sign invariant above* $\hat{T}$ if it is either invertible or zero modulo $\hat{T}$.

Thus, if $q \in \mathbb{R}[\mathbf{x}]$, then $p = q \mod \hat{T}$ if $V_p \cap Z(\hat{T}) = V_q \cap Z(\hat{T})$, that is, you cannot distinguish them just by looking at $\hat{T}$.

In the special case where $\hat{T} = S$ is a cylindrical system, we make the following definition:

Definition 3.6 (GCD mod S)**.** $g \in \mathbb{R}[\mathbf{x}]$ is a GCD of p and f mod S if $g(\alpha)$ is a GCD of $p(\alpha)$ and $f(\alpha) \in \mathbb{R}[z_n]$, for any $\alpha \in Z(S)$.

It seems likely that the bulk of the work in the algorithm involves computing the GCDs.

Since we are working over an algebraically closed field, Definition 3.4 is introduced for the following reason:

If p_1 and p_2 are univariate polynomials over an algebraically closed field, then they are coprime if and only if they do not ever both vanish at the same place. This explains the word "separates": If we look at the fibre above α: It is a vertical line with points marked on it, which are the zeroes of all the $p \in P$. We are requiring that these finitely many points should all be different, i.e. the set of zeroes of $p_1(\alpha, x_n)$ and $p_2(\alpha, x_n)$ are disjoint.

Now as α varies in $Z(S)$, we look at the loci where $p_i(\alpha, x_n)$ are zero, then "separates above" means informally that these loci do not run into one another.

In the base case $n = 1$, each D_i for $1 \leq i \leq r$ is the vanishing locus of p_i and D_{r+1} is everything else, and the p_i are coprime and squarefree.

Thus, $p_i = \prod_{w_j \in D_i}(z - w_j)$, and these are squarefree by construction. The p_i being coprime means D_1 through D_r are disjoint.

The main constraint is that the p_i have to be real polynomials. In other words, the symmetric polynomials in the elements w_j of each D_i have to be real. For example, if D_1 consists of one point it must be real: if it is two points, they have to be complex conjugates so their sun and product are real.

For $n > 1$, inductively, we have a cylindrical decomposition of $\mathbb{C}^{n-1}$. For each of the D_i we choose some polynomials that separate above D_1. The line above α in D_i contains points where these polynomials vanish, because they have been chosen to separate above D_1, the polynomials are squarefree and coprime, and we repeat the process on the fibre.

We continue making the tree in the same way, adding leaves $D_{1,1}$ to D_{1,r_1+1}. Each of these is associated with either a constraint (a polynomial p_i or $p_{i,j}$ and so on) or with no constraints. Each cell $D_{i,1}, \ldots, D_{i,\ell}$ is the zero set of a cylindrical system induced by T.

3.3 Construction of a cylindrical tree

A brief overview is shown in the meta-algorithm Algorithm 1.

The actual construction of a cylindrical tree, as outlined above, involves many algorithms that call each other, as seen in [CMM12]. A brief overview is shown in Algorithm 1.

Algorithm 1 `CylindricalDecompose`($\mathcal{F}$) Meta-algorithm

Input: $\mathcal{F}$ a set of non-constant polynomials in $\mathbb{R}[\mathbf{x}]$
Output: An $\mathcal{F}$-invariant cylindrical decomposition of $\mathbb{C}^n$

`CylindricalDecompose`($\mathcal{F}$)
- `Intersect`$_n(p, T)$
  - `IntersectPath`$_n(p, \Gamma, T)$
    - `IntersectMain`$_k(p, \Gamma_k, T_k)$
      - `Squarefree`$_k(p, \Gamma_{k-1}, T_{k-1})$
        - `MakeLeadingCoefficientInvertible`$_k(p, p, \Gamma_{k-1}, T_{k-1})$
          - `IntersectPath`$_{k-2}(\mathrm{lc}(\bar{p}, x_{k-1}), \Gamma_{k-1}, T_{k-1})$
            - $\ldots$
        - `Gcd`$_k(f, \mathrm{sep}(f), C_{k-1}, T_{k-1})$
          - `Gcd`$_k(f, \mathrm{sep}(f), S, d, 0, C_{k-1}, T_{k-1})$
            - `IntersectPath`$_{k-1}(s_i, C_{k-1}, T_{k-1})$
              - $\ldots$
        - `Gcd`$_k(sp, f, C_{k-1}, T_{k-1})$
          - `Gcd`$_k(sp, f, S, d, 0, C_{k-1}, T_{k-1})$
            - `IntersectPath`$_{k-1}(s_i, C_{k-1}, T_{k-1})$
        - `Cofactor`$(sp, L.Gcd[sp, f], f)$

In Algorithm 1 we refer to:

- $\mathcal{F}$ a set of non-constant polynomials in $\mathbb{R}[\mathbf{x}]$
- T a tree with level n
- p a non-constant polynomial in $\mathcal{F}$ with level k
- Γ a path of T with level n
- C a path derived from of Γ after spitting, with level n
- T_k, Γ_k and C_k truncations of T, Γ and C to level k ($T_n = T, \Gamma_n = \Gamma$, $C_n = C$)
- V a leaf (of Γ or C)
- sp, the squarefree part of p
- cp, gg, cf the outputs of $\texttt{CoFactor}(p, g, f)$
- L the parent of V
- S the subresultant chain of p and f
- d the GCD degree
- s_i the principal subresultant coefficient

(One could consider calling these inside-out instead of outside-in.)

We describe briefly each of these sub-algorithms.

- Top-level algorithm $\texttt{CylindricalDecompose}(\mathcal{F})$ takes a set $\mathcal{F}$ of non-constant polynomials in $\mathbb{R}[\mathbf{x}]$ and outputs an $\mathcal{F}$-invariant cylindrical decomposition of $\mathbb{C}^n$.

  It first creates the tree T with only on vertex V_0, the root r of T. It then constructs, for $1 \leq i \leq n$, the vertex V_i, with the attached formula *"any* x_i*"*, where V_i is the child of the vertex V_{i-1}. Then, for each $p \in \mathcal{F}$, it calls $\texttt{Intersect}_n(p, T)$.

- $\texttt{Intersect}_n(p, T)$ takes this polynomial p and tree T and outputs a refined cylindrical decomposition such that p is sign-invariant above each path of T. For each path Γ in T in some fixed traversal order, it then calls the algorithm $\texttt{IntersectPath}_n(p, \Gamma, T)$.

- $\texttt{IntersectPath}_n(p, \Gamma, T)$ takes the cylindrical tree T, the path Γ of T and the polynomial p and returns a refined cylindrical decomposition T such that p is sign-invariant above each path derived from Γ. If p is constant, we are done. If it is not, it checks the level k of p. If $k = n$, it calls $\texttt{IntersectMain}_n(p, \Gamma, T)$, otherwise it defines T_k and Γ_k as the "truncations" of T and Γ to level k, (that is, nodes higher than level k are ignored and so the level k nodes are the leaves), then calls $\texttt{IntersectMain}_k(p, \Gamma_k, T_k)$, updates the path and for each leaf V of Γ and attaches needed information.

- $\texttt{IntersectMain}_n(p, \Gamma, T)$ is the main algorithm, refining the cylindrical tree T into a cylindrical decomposition such that p is sign-invariant above each path

derived from Γ. It does this by first defining T_{n-1} and Γ_{n-1} as above, then calls $\texttt{Squarefree}_n(p, \Gamma_{n-1}, T_{n-1})$ and updates the path. For each $C \in \Gamma$, it sets V as the leaf of C and C_{n-1} as the truncation of C to level $n-1$. It then sets sp as the squarefree part of p on the leaf of C_{n-1}.

- If $sp = 0$ or $sp = 1$, it sets the sign of V to 0 or 1 accordingly (generally, we say the sign of 0 in $\mathbb{C}$ is 0, and any other complex number is 1).
- If V is of the form *"any x_n"* then it is split into two new vertices V_1 and V_2, where V_1 has formula "$sp = 0$" and V_2 has formula "$sp \neq 0$" and everything else is unchanged. These two nodes are the leaves and are children of C_{n-1}.
- If V is of the form $f = 0$ or $f \neq 0$, $\texttt{Gcd}_n(sp, f, C_{n-1}, T_{n-1})$ is called and the path is updated. For each leaf V of C and its parent L, it calls $\texttt{CoFactor}(sp, L.Gcd[sp, f], f)$ to obtain cp, g and cf.
  (a) If V is of the form $f = 0$, then if $g = 1$, it sets V's sign over p as 1, if not, and if $cf = 1$, it is set as 0. Otherwise V is split into two vertices V_1 and V_2, where V_1's formula is $g = 0$ and V_2's formula is $cf = 0$, and these are now the children of L.
  (b) If V is of the form $f \neq 0$, then if $cp = 1$ its sign is set to 1, otherwise V is split into V_1 and V_2, with V_1 of the form $cp = 0$ with sign 0 at p and V_2 of the form $(f * cp) \neq 0$ and sign 1 at p, and both inheriting everything else from V, and becoming children of L.

- $\texttt{Squarefree}_n(p, \Gamma, T)$ returns p^* for the polynomial p of level n, with the properties that $p = p^*$ modulo C, if p^* is level n then both $\text{init}(p^*)$ and the descriminant $\text{disc}(p^*)$ are invertible mod C, and if it is of level less than n, it is either 0 or 1.

  If $n = 1$, then it is just the squarefree part of the root node. Otherwise, it calls $\texttt{MakeLeadingCoefficientInvertible}_n(p, p, \Gamma, T)$ and for a path C, it sets p with invertible leading coefficient as f, then if this has a level less than n or has degree 1 in x_n, the squarefree part of p is simply f.

  Otherwise, it calls $Gcd_n(f, \text{sep}(f), C, T)$ and for each leaf L of C sets this as g. If $g = 1$ then the squarefree part of p is f, otherwise it is $\text{pquo}(f, g)$.

- $\texttt{Gcd}_n(p, f, \Gamma, T)$ takes T, Γ, p and f a polynomial of level n whose initial is invertible modulo Γ, and sets up so that $Gcd[p, f]$ is a GCD of p and f modulo C.

  The algorithm does this by setting S as the subresultant chain of p and f [Jir95], and if $\text{mdeg}(p) \geq \text{mdeg}(f), d = \text{mdeg}(f)$, otherwise it is $\text{mdeg}(p) + 1$. Then $\texttt{Gcd}_n(p, f, S, d, 0, \Gamma, T)$ is called.

- For $\texttt{Gcd}_n(p, f, S, d, i, \Gamma, T)$, i is a non-negative integer such that $0 \leq i \leq d$ and for all $0 \leq j \leq i$, the principal subresultant coefficient s_j [Jir95] is zero modulo

Γ. This algorithm produces a refined cylindrical tree T such that above each path C, $Gcd[p, f]$ is a GCD of p and f modulo C.

If $i = d$ then $Gcd[p, f]$ in Γ is S_i. Otherwise, call $\texttt{IntersectPath}_{n-1}(s_i, \Gamma, T)$. If the sign of s_i in C is 1, then if $i = 0$, the GCD of p and f is 1, if it isn't, it is S_i. If the sign of s_i is 0, then call $\texttt{Gcd}_n(p, f, S, d, i + 1, C, T)$

- `CoFactor(p,g,f)` takes two level n polynomials p and f and a polynomial g which is either level n or is 1 and returns cp, gg and cf:
  - If $g = 1$, then $cp = p, gg = 1, cf = f$.
  - If $\text{mdeg}(g) = \text{mdeg}(f) = \text{mdeg}(p)$, then $cp = 1, gg = f, cf = 1$.
  - If $\text{mdeg}(g) = \text{mdeg}(f) \neq \text{mdeg}(p)$, then $cp = \text{pquo}(f, gg), gg = f, cf = 1$.
  - If $\text{mdeg}(g) = \text{mdeg}(p) \neq \text{mdeg}(f)$, then $cp = 1, gg = p, cf = \text{pquo}(p, gg)$.
  - Otherwise, $cp = \text{pquo}(p, g), gg = g, cf = \text{pquo}(f, g)$.
- $\texttt{MakeLeadingCoefficientInvertible}_n(p, \bar{p}, \Gamma, T)$ takes a tree T, a path Γ, a polynomial p and a polynomial $\bar{p}$ such that $p = \bar{p}$ modulo Γ and outputs, above each path C of T derived from Γ, a polynomial p^* such that $p = p^*$ modulo Γ and if p^* is level n, then $\text{init}(p^*)$ is invertible modulo C, and if a lower level, p^* is either 0 or 1.

  It does this by calling $\texttt{IntersectPath}_{n-1}(\text{lc}(\bar{p}, x_n), \Gamma, T)$, and for each C, checks the sign of $\text{lc}(\bar{p}, x_n)$: if it is 1 and the level of $\bar{p}$ is less than n, then $p^* = 1$, if it is 1 and the level of $\bar{p}$ is n, then $p^* = \bar{p}$. If the sign is 0 and the level is less than n, then $p^* = 0$. If the sign is 0 and the level is n, it calls $\texttt{MakeLeadingCoefficientInvertible}_n(p, \text{tail}(\bar{p}), C, T)$

3.4 Building a CAD from a Complex Cylindrical Tree

The final step is to compute a CAD of $\mathbb{R}^n$ from a cylindrical decomposition of $\mathbb{C}^n$.

Suppose $p \in \mathbb{R}[\mathbf{x}]$ and S is a connected semi-algebraic subset of $\mathbb{R}^{n-1}$. We say that p is *delineable* on S if the real zeros of p define continuous semi-algebraic functions $\theta_1, \ldots, \theta_s$ such that, for all $\alpha \in S$ we have $\theta_1(\alpha) < \cdots < \theta_s(\alpha)$. In other words, p is delineable on S if its real zeroes determine a stack over S.

Applying this inductively to our cylindrical tree yeilds an $\mathcal{F}$-invariant CAD of $\mathbb{R}^n$.

3.5 Complexity of RC-CAD

RC-CAD is a different algorithm to PL-CAD at a much earlier stage of development. It has been implemented in *Maple*, but not a very great deal has been done on trying to improve its performance. There are some experiments and timings conducted by Chen and Moreno Maza, which suggest it is at least competitive with PL-CAD sometimes. However, there are no theoretical complexity estimates have been done, and as a consequence we have no idea of when it performs well or badly.

For PL-CAD, on the other hand, there have been serious attempts to understand the complexity in detail (for example [Nai21, Ton21, BDE+16] and the references therein) and also to understand where the worst cases occur.

4 Current and Future Research

Using the meta-algorithm described above as the first step, I plan to carry out a detailed complexity analysis of RC-CAD, and once I have complexity estimates, refine them and see if I can determine cases where complexity is better or worse and get some kind of heuristic for when it is better to use RC-CAD or PL-CAD. To help with this I have also been looking at the *Maple* source.

I also intend to doing some experiments on some further test cases (such as the Piano Mover's Problem in [Dav86], where PL-CAD fails).

Given there are competing versions of CAD, various desirable properties that a CAD might have such as being well-based [DLS19], we would like to know what kind of CAD RC-CAD produces, and whether we can we "coerce" it to produce the types of CAD we want.

Beyond that, there is another algorithm using Comprehensive Gröbner Systems (CGS-CAD) that is not implemented as far as we know, so there are no timings and very little knowledge of complexity. This is something we could also investigate in the future.

References

[ACM84] Dennis Arnon, George Collins, and Scott Mccallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM J. Comput.*, 13:865–877, 11 1984.

[ACM98] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: The basic algorithm. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 136–151, Vienna, 1998. Springer Vienna.

[BD02] Russell Bradford and James H. Davenport. Towards better simplification of elementary functions. In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 16–22, New York, NY, USA, 2002. Association for Computing Machinery.

[BDE+16] Russell Bradford, James H. Davenport, Matthew England, Scott McCallum, and David Wilson. Truth table invariant cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 76:1–35, 2016.

[Bro] Christopher W Brown. Q e p c a d - quantifier elimination by partial cylindrical algebraic decomposition. https://www.usna.edu/CS/qepcadweb/B/QEPCAD.html. Accessed: 2022-10-21.

[Bro01] Christopher W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.

[CMM] Changbo Chen and Marc Moreno Maza. The regularchains library. https://www.regularchains.org/. Accessed: 2022-10-21.

[CMM12] Changbo Chen and Marc Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *CoRR*, abs/1210.5543, 2012.

[CMMXY09] Changbo Chen, Marc Moreno Maza, Bican Xia, and Lu Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, page 95–102, New York, NY, USA, 2009. Association for Computing Machinery.

[Col75] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Lecture Notes in Computer Science*, 1975.

[Dav86] James Davenport. A "piano movers" problem. *ACM Sigsam Bulletin*, 20:15–17, 02 1986.

[DH88] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1):29–35, 1988.

[DLS19] J. H. Davenport, A. F. Locatelli, and G. K. Sankaran. Regular cylindrical algebraic decomposition. *Journal of the London Mathematical Society*, 101(1):43–59, Jul 2019.

[Eng] M. England. The projectioncad package. https://matthewengland.coventry.domains/ProjectionCAD.html. Accessed: 2022-10-21.

[Jir95] M. Jirstrand. *Cylindrical Algebraic Decomposition: An Introduction.* LiTH-ISY-R. Linköpings university, 1995.

[Laz94] D. Lazard. *An Improved Projection for Cylindrical Algebraic Decomposition*, pages 467–476. Springer New York, New York, NY, 1994.

[McC84] S. McCallum. *An Improved Projection Operation for Cylindrical Algebraic Decomposition.* University of Wisconsin–Madison, 1984.

[Nai21] Akshar Nair. *Curtains in Cylindrical Algebraic Decomposition.* PhD thesis, University of Bath, 2021.

[Red] Redlog. Redlog. http://www.redlog.eu/. Accessed: 2022-10-21.

[Res] Wolfram Research. Wolfram mathematica: Modern technical computing. http://www.wolfram.com/mathematica/. Accessed: 2022-10-21.

[SS83a] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem I. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36(3):345–398, 1983.

[SS83b] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem. II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.

[Ton21] Zak Tonks. *Poly-algorithmic Techniques in Real Quantifier Elimination*. PhD thesis, University of Bath, 2021.

[WDEB13] David Wilson, James H. Davenport, Matthew England, and Russell Bradford. A "piano movers" problem reformulated. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE, sep 2013.

[YA07] Hitoshi Yanami and Hirokazu Anai. Synrac: A maple toolbox for solving real algebraic constraints. *ACM Commun. Comput. Algebra*, 41(3):112–113, sep 2007.