On algorithms for real algebraic curves

F. Cucker
F. Roselló

Report LSI–89–24

Abstract: In this note we give two new algorithms for computing a cylindrical algebraic decomposition as well as the topological type of a real algebraic curve that run in time $O(n^{16}\log^3 n)$ and $O(n^{19}\log^3 n)$ respectively, improving then the running time of the best algorithms known so far for the same problems.

Resum: En aquesta nota donem dos nous algorismes per a computar una descomposició algebraica cilíndrica i el tipus topològic d'una corba algebraica real amb complexitat $O(n^{16}\log^3 n)$ i $O(n^{19}\log^3 n)$ respectivament. Aquests algorismes milloran així el temps de càlcul dels algorismes més ràpids coneguts fins ara per a resoldre els mateixos problemes.

On algorithms for real algebraic curves

F. Cucker †
Dept. L.S.I.
Facultat d'Informàtica
Barcelona 08028
SPAIN

F. Rosselló ‡
Dept. Àlgebra i Geometria
Universitat de Barcelona
Barcelona 08007
SPAIN

During the last years several researchers have considered the problem of finding polynomial–time sequential algorithms for the computation of the topology of a real algebraic plane curve. Such algorithms have been given for instance in [1] and [4], both by coding real algebraic numbers using isolating intervals and restricted to non–singular curves, and in [6], using coding *à la Thom* and applying it to any curve.

Let $F \in \mathbb{Z}[X,Y]$ and let $\mathcal{C} \subset \mathbb{R}^2$ be the algebraic set defined by the equation $F(X,Y) = 0$. Let p be the degree of F and, if $F = \sum a_{i,j}X^iY^j$, then let $|F| = \log\sqrt{\sum a_{i,j}^2}$ and $n = max\{p, |F|\}$.

With these notations, the algorithm given in [1] is shown to run in time $O(n^{30})$, while no complexity estimates are given for the one in [4]. As far as the algorithm in [6] goes, in [8] it is proved that it runs in time $O(n^{28})$, and also that a cylindrification of $\mathcal{C}$ can be obtained in time $O(n^{23})$.

Recently, it has been noted that the ground algorithm used to code real algebraic numbers *à la Thom* has a complexity smaller than the one used for the complexity estimates given in [8]. The new bounds for the complexity of such ground algorithm will appear in [7].

Using these new bounds it can be shown that the cylindrification algorithm in [6] runs actually in time $O(n^{17}(\log n)^3)$ while the one computing the topological type of the curve runs in $O(n^{21}(\log n)^4)$.

In this note we introduce some modifications to the latter algorithms, obtaining a cylindrification algorithm which runs in time $O(n^{16}(\log n)^3)$ and an algorithm for the computation of the topological type which runs in $O(n^{19}(\log n)^3)$. The ground tools for our algorithms remain Sturm–Habicht sequences and coding *à la Thom*. On the other hand, they differ from those given in [6] and [8] by the fact that we systematically use real algebraic numbers to get the information which in the quoted algorithms is obtained by working with infinitesimals.

1. Ground tools.

1.1. Sturm–Habicht sequences and systems of equalities and inequalities.

We begin by introducing some notation and terminology that will be useful for the rest of this note. In the sequel we shall call ***sign condition***, or simply ***sign***, one of the following three: < 0, > 0 and $= 0$, and we shall denote them by $-, +$ and 0 respectively. We shall call ***generalized sign condition*** any sign condition as well as ≤ 0 and ≥ 0.

Given two k–tuples of sign conditions $\epsilon = (\epsilon_1, \ldots, \epsilon_k)$ and $\epsilon' = (\epsilon'_1, \ldots, \epsilon'_k)$, we shall say that ϵ is ***compatible*** with ϵ' when

$$\epsilon_i \neq \epsilon'_i \implies \epsilon_i = 0, \qquad i = 1, \ldots, k.$$

Given a polynomial $F = \sum a_{ij}X^iY^j \in \mathbb{Z}[X,Y]$ we recall that its ***norm*** is defined as $\sqrt{\sum a_{ij}^2}$, and that its ***size*** $|F|$ is defined as the logarithm of its norm.

Now, let A be an ordered domain, K its field of fractions and $\overline{K}$ the real closure of K. Given polynomials $P, Q_1, \ldots, Q_k \in A[X]$ and a k–tuple of sign conditions $\epsilon = (\epsilon_1, \ldots, \epsilon_k)$, we denote by $c_\epsilon(P; Q_1, \ldots, Q_k)$ the number of solutions in $\overline{K}$ of the system

$$\begin{aligned} P(X) &= 0 \\ Q_1(X)\ &\epsilon_1 \\ &\vdots \\ Q_k(X)\ &\epsilon_k \end{aligned}$$

We shall say that ϵ is satisfied when $c_\epsilon(P, Q_1, \ldots, Q_k) \neq 0$.

† Partially supported by DGICyT PB 860062 and the ESPRIT BRA Program of the EC under contract no. 3075, project ALCOM.
‡ Partially supported by DGICyT PB 870137.

In [5] a new tool for computing the c_ϵ's is introduced: the Sturm–Habicht sequences. These sequences are based on the general theory of subresultants and have the following features:

— they yield the same information as the generalized Sturm sequences
— they can be computed in polynomial time
— they have good specialization properties

We don't pretend to introduce here this tool; the interested reader can get acquainted with it by looking at [5]. We just recall that if $F, G \in A[Y]$ have degree n and m respectively, then the Sturm–Habicht sequence associated to F and G is a sequence of polynomials $F_{s+1} = F, F_s = G, F_{s-1}, \ldots, F_0$ in $A[Y]$, where $s = n$ when $n > m$ and $s = m + 1$ otherwise. Each polynomial F_j in this sequence has degree $d_j \leq j$. One calls the ***j–th Sturm–Habicht principal coefficient*** of F and G the coefficient of degree j of F_j, $j = 0, \ldots, s+1$. Some of these Sturm–Habicht coefficients may vanish.

Moreover, if $A = \mathbb{Z}[X]$ and j is the smallest index such that $F_j \neq 0$, then F_j agrees, up to an integer factor, with the g.c.d. of F and G.

If we denote now by $Sth(P, Q)$ the difference of sign variations taken by the Sturm–Habicht sequence associated to P and $P'Q$ at $-\infty$ and $+\infty$, then we have the following

Theorem 1.1. (see [5] and [8])

Let $P, Q \in A[Y]$ be arbitrary polynomials of degrees p and q respectively.

i) In the Sturm–Habicht sequence associated to P and Q appear at most $\min\{p, q\} + 3$ non–zero polynomials.

ii) The Sturm–Habicht sequence of P and Q can be computed in $O(pq)$ arithmetic operations over A. Moreover, if $A = \mathbb{Z}[X]$ and $\overline{p}$ and $\overline{q}$ denote the degrees of P and Q with respect to the variable X, then the sizes of the coefficients appearing in the Sturm–Habicht sequence of P and Q are bounded by $O(p|Q| + q|P| + (p+q)\log(\overline{p}+\overline{q}))$, and their degrees are at most $\overline{p}q + p\overline{q}$.

iii) $Sth(P, Q) = c_+(P; Q) - c_-(P; Q)$.

iv) $Sth(P, Q)$ can be obtained by looking at the signs of the Sturm–Habicht principal coefficients of P and $P'Q$. ∎

This theorem allows us to compute the values of $c_0(P; Q)$, $c_+(P; Q)$ and $c_-(P; Q)$ for $P, Q \in \mathbb{Z}[X]$. We just have to solve the following matrix equality

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_0(P;Q) \\ c_+(P;Q) \\ c_-(P;Q) \end{bmatrix} = \begin{bmatrix} Sth(P,1) \\ Sth(P,Q) \\ Sth(P,Q^2) \end{bmatrix}.$$

The SI procedure, which we now describe, allows us to get the c_ϵ's when several polynomials Q's are involved. For details about it see [7].

Procedure SIadd.

The ***input*** of this procedure is the following set of data
— polynomials $P, Q_1, \ldots, Q_k$ with integer coefficients
— the list of satisfied k–tuples of sign conditions $\epsilon_1, \ldots, \epsilon_{r(k)}$
— the vector c_k of non–zero values $c_{\epsilon_1}(P; Q_1, \ldots, Q_k), \ldots, c_{\epsilon_{r(k)}}(P; Q_1, \ldots, Q_k)$
— polynomials $R_1, \ldots, R_{r(k)}$ which are products of some of the Q's
— a vector v_k whose elements are the values $Sth(P, R_j)$, $j = 1, \ldots, r(k)$
— an invertible $r(k) \times r(k)$ matrix $A(k)$ satisfying the equation $A(k) \cdot c_k = v_k$
as well as a new polynomial Q_{k+1}.

Its ***output*** is
— a list containing $P, Q_1, \ldots, Q_{k+1}$
— the list of satisfied $(k+1)$–tuples of sign conditions $\epsilon_1, \ldots, \epsilon_{r(k+1)}$
— the vector c_{k+1} of non–zero values $c_{\epsilon_1}(P; Q_1, \ldots, Q_{k+1}), \ldots, c_{\epsilon_{r(k+1)}}(P; Q_1, \ldots, Q_{k+1})$
— polynomials $R_1, \ldots, R_{r(k+1)}$ which are products of some of the Q's
— a vector v_{k+1} whose elements are the values $Sth(P, R_j)$, $j = 1, \ldots, r(k+1)$
— an invertible $r(k+1) \times r(k+1)$ matrix $A(k+1)$ satisfying the equation $A(k+1) \cdot c_{k+1} = v_{k+1}$

The procedure performs the following steps:

1) Let $R_{r(k)+j} = R_j Q_{k+1}^2$ for $j = 1, \ldots, r(k)$ and $R_{2r(k)+j} = R_j Q_{k+1}$ for $j = 1, \ldots, r(k)$. We compute the $3r(k)$–dimensional vector v whose j–component is the number $Sth(P, R_j)$.
2) We now define a list of $3r(k)$ $(k+1)$–tuples of sign conditions whose first $r(k)$ elements are the input k–tuples followed by 0, the second $r(k)$ elements are the same k–tuples followed by $+$, while for the last $r(k)$ elements we add a $-$.
3) We compute the Kronecker product A of $A(k)$ with

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

i.e. the matrix

$$\begin{bmatrix} A(k) & A(k) & A(k) \\ 0 & A(k) & A(k) \\ 0 & A(k) & -A(k) \end{bmatrix}$$

4) We compute a $3r(k)$–dimensional vector c satisfying the equation $A \cdot c = v$
5) We get a new vector c_{k+1} by deleting the zero components of c, a new matrix A' by deleting the columns corresponding to these components in A and a new list of $r(k+1)$ $(k+1)$–tuples of sign conditions by deleting the ones which are not satisfied.
6) We get an invertible square submatrix $A(k+1)$ of A' by conserving the first $r(k+1)$ linearly independent rows, and a new vector v_{k+1} by conserving in v the components corresponding to those rows.

Procedure SI.

The ***input*** of this procedure consists of a sequence of polynomials with integer coefficients $P, Q_1, \ldots, Q_k$.

Its ***output*** is a list with
— the list of satisfied k–tuples of sign conditions $\epsilon_1, \ldots, \epsilon_{r(k)}$
— the vector c_k of non–zero values $c_{\epsilon_1}(P; Q_1, \ldots, Q_k), \ldots, c_{\epsilon_{r(k)}}(P; Q_1, \ldots, Q_k)$
— polynomials $R_1, \ldots, R_{r(k)}$ which are products of some of the Q's
— a vector v_k whose elements are the values $Sth(P, R_j)$, $j = 1, \ldots, r(k)$
— an invertible $r(k) \times r(k)$ matrix $A(k)$ satisfying the equation $A(k) \cdot c_k = v_k$

The procedure consists in performing k times the SIadd procedure.

Lemma 1.1. ([7])
Let r be the number of real roots of P. The polynomials R_i $(i = 1, \ldots, r(k))$ in the output of $SI(P; Q_1, \ldots, Q_k)$ are products of at most $\log r$ polynomials Q_j or Q_j^2 $(j = 1, \ldots, k)$. ∎

Notice also that $r(k) \leq r$.

Proposition 1.1.
Let r be the number of roots of P, d a bound on the degrees of the polynomials $P, Q_1, \ldots, Q_k$, and N a bound on their sizes. The procedure SI runs in time $O(krd^4(\log r)^3 N^2)$.

Proof.
We perform k times the procedure SIadd. In the first step of each performance, we compute at most $2r$ Sturm–Habicht sequences associated to P and polynomials R which are products of P' and at most $O(\log r)$ polynomials Q_i $(i = 1, \ldots, k)$. So, the degree of such a polynomial R is at most $O(d \log r)$ and its size is at most $O(N \log r)$. It follows that the sizes of the coefficients of the corresponding Sturm–Habicht sequences are bounded by $O(dN \log r)$. Thus, each one of these sequences costs $O(d^4(\log r)^3 N^2)$. Since the cost of the remaining steps of the procedure SIadd is negligible, we get the stated total bound by multiplying the latter cost by kr. ∎

For the sake of simplicity, in the sequel we shall bound the number of real roots of a polynomial $P \in \mathbb{Z}[X]$ by its degree. Notice that, in average, the number of real roots is much smaller than the degree (see [3]).

1.2. *Real algebraic numbers.*

A way of coding real algebraic numbers is given in [2] which relies upon the preceding algorithms. It also relies on the following result

Thom's lemma. ([2])

Let $P \in A[X]$ be a polynomial of degree n and $\epsilon = (\epsilon_0, \ldots, \epsilon_n)$ a n–tuple of generalized sign conditions. Let

$$A(\epsilon) = \{x \in \overline{K} \mid P^{(i)}(x)\epsilon_i,\ i = 0, \ldots, n\}.$$

Then: i) $A(\epsilon)$ is either empty or semialgebraically connected; ii) if $A(\epsilon)$ is not empty, then its closure is $A(\underline{\epsilon})$ where $\underline{\epsilon}$ is obtained from ϵ by relaxing the sign conditions. ∎

Seting $\epsilon_0 = 0$ we get a way of individualizing the roots of P and then a way of coding real algebraic numbers.

Proposition 1.2. (Coste and Roy.)

Let $P \in A[X]$ be a polynomial of degree $n \geq 1$.

i) Let $\epsilon_1, \ldots, \epsilon_n$ be sign conditions. Then the system

$$\begin{aligned} &P(X) = 0 \\ &P'(X)\ \epsilon_1 \\ &\quad\vdots \\ &P^{(n)}(X)\ \epsilon_n \end{aligned}$$

has at most one solution in $\overline{K}$.

ii) Let ξ_1 and ξ_2 be two (different) elements of $\overline{K}$, and set $\epsilon_j = \operatorname{sign} P^{(j)}(\xi_1)$, $\delta_j = \operatorname{sign} P^{(j)}(\xi_2)$ $(j = 0, \ldots, n)$. If the $(n+1)$–tuples of sign conditions $(\epsilon_0, \ldots, \epsilon_n)$ and $(\delta_0, \ldots, \delta_n)$ are different, then we can deduce from them the relative position of ξ_1 and ξ_2 in the following way:

Let j be the smallest number such that $\epsilon_{n-j} \neq \delta_{n-j}$. It is clear that $j \geq 1$ and that $\epsilon_{n-j+1} = \delta_{n-j+1}$ is different from 0.

a) If $\epsilon_{n-j+1} > 0$ then $\xi_1 > \xi_2$ iff $P^{(n-j)}(\xi_1)$ is greater than $P^{(n-j)}(\xi_2)$, and we can decide it by looking at ϵ_{n-j} and δ_{n-j}.

b) If $\epsilon_{n-j+1} < 0$ then $\xi_1 > \xi_2$ iff $P^{(n-j)}(\xi_1)$ is smaller than $P^{(n-j)}(\xi_2)$, and we can decide it by looking at ϵ_{n-j} and δ_{n-j}. ∎

Remark 1.1.

This Proposition allows us to code each root of P by the n–tuple of signs taken on it by the derivatives of P. Notice that from these codes we can deduce the relative position of all these roots, as it is shown in point (ii) of the last Proposition.

We want also to remark that we can always sort a root ξ of P and another real number ζ by comparing the code $(\epsilon_1, \ldots, \epsilon_n)$ of ξ (as a root of P) and the $n+1$–tuple of signs taken by P and its derivatives on ζ (if $\zeta \neq \xi$, then this $(n+1)$–tuple of signs is different from $(0, \epsilon_1, \ldots, \epsilon_n)$). This observation allows one to simplify the algorithm for the comparison of two real algebraic numbers given in [7]. ∎

Procedure RAN.

Its ***input*** is a polynomial P with integer coefficients.

Its ***output*** is the list of n–tuples of sign conditions satisfied by the derivatives of P on its roots, together with a matricial equation like in SI. The codes are sorted in increasing order of the corresponding roots.

If n is the degree of P, then the algorithm performs the procedure SI$(P; P^{(n-1)}, \ldots, P')$.

Procedure RANI.

Its ***input*** is the output of $RAN(P)$ and a new polynomial Q.

Its ***output*** is the sign Q takes on the roots of P.

The algorithm just applies SIadd to the output of $RAN(P)$ and Q.

Proposition 1.3.

i) Let d and N be the degree and the size of P, respectively. Then RAN(P) runs in time $O\big(d^6(\log d)^3(d+N)^2\big)$.

ii) If now Q is a polinomial of degree q and size S, the computation RANI(RAN(P), Q) takes time $O\big(d^2(q+d\log d)(d(d+N)\log d+dS+qN)^2\big)$.

Proof.

i) The size of $P^{(i)}/i!$ is bounded by $i+N$, and thus the sizes of the coefficients in the Sturm–Habicht sequences are bounded by $O(d(d+n)\log d)$. Now, similar arguments to those used to prove Proposition 1.1 give us a total time bound in $O(d^6(\log d)^3(N+d)^2)$.

ii) As in Proposition 1.1, the dominant step is the first one, in which we compute at most $2d$ Sturm–Habicht sequences of P and products $P'RQ$, where R is a product of at most $O(\log d)$ derivatives of P. The degree of each $P'RQ$ is bounded by $O(q+d\log d)$ and its size by $O((d+N)\log d+S)$.

Each Sturm–Habicht sequence then takes time $O\big(d(q+d\log d)(d(d+N)\log d+dS+qN)^2\big)$, so we get the stated total computation time. ∎

1.3. Specialized computations.

As far as we are concerned in this note, two main procedures are needed for specialized computations. They respectively count and code the roots of a bivariate polynomial one of whose indeterminates is specialized at a root of a given integer polynomial. For details about them, the reader can look up [8].

So, let $F \in \mathbb{Z}[X,Y]$ and $D \in \mathbb{Z}[X]$, and let $\xi_1,\dots,\xi_r$ denote the real roots of D, that we shall suppose coded by RAN.

Procedure $\mathbf{St_1}$.

Its ***input*** is the pair (T,S) where
— T is the output of RAN(D).
— S is the Sturm–Habicht sequence of F and F'_Y

Its ***output*** is the sequence $n_1,\dots,n_r$, where n_j is the number of real roots of $F(\xi_j,Y)$.

Since each n_j coincides with $Sth(F(\xi_j,Y),1)$ and this number can be deduced from the Sturm–Habicht principal coefficients of $F(\xi_j,Y)$ and $F'_Y(\xi_j,Y)$, the procedure just performs RANI(T,c), for every such a principal Sturm–Habicht coefficient c.

We shall denote by $\mathrm{St}_1(D,F)$ the invocation $\mathrm{St}_1(T,S)$.

The procedure $\mathbf{RAN_1}$.

Its ***input*** is the pair $((D,\xi_j),F)$ where (D,ξ_j) denotes the output of RAN applied to D as well as the choice of a particular root ξ_j of D.

Its ***output*** is a data structure like the one produced by RAN, but with bivariated polynomials instead of univariated ones. The collection of sign conditions codes now the roots of $F(\xi_j,Y)$.

The procedure is based on the same principle as RAN. The only difference is that, in the present case, the Sturm–Habicht queries, which form the vector appearing as the right member of the matricial equality, must be computed in the same way as the values $Sth(F(\xi_j,Y),1)$ in St_1.

Remark 1.2.

Notice that, while St_1 computes its output for all the roots of D simultaneously, this is not the case with RAN_1, where a single root must be distinguished. The reason is that, while each specialized Sturm–Habicht query of the form $Sth(F,R)$ reduces to several applications of RANI for the principal coefficients of the sequence and this procedure is executed simultaneously for all the roots of D, the forking process that generates the products of derivatives to be used in these queries varies with each root. ∎

As far as the running time of these procedures goes, we have the following result.

Proposition 1.4.
Let d and N be the degree and the size of $D \in \mathbb{Z}[X]$ respectively. Likewise, let p and M be the total degree and the size of $F \in \mathbb{Z}[X,Y]$ respectively. We have

i) $\mathrm{St}_1(D,F)$ runs in time

$$O\big(pd^2(p^2+d\log d)(d(d+N)\log d + dp(M+\log p)+p^2N)^2\big).$$

ii) Let ξ be a root of D. Then $\mathrm{RAN}_1((D,\xi),F)$ runs in time

$$O\big(p^{10}(\log p)^5(p+M)^2 + p^3d^2(p^2\log p + d\log d)(d(d+N)\log d + dp(p+M)\log p + p^2N\log p)^2\big)$$

Proof.
i) We must call RANI(RAN(D), c), for every principal Sturm–Habicht coefficient c in the input sequence. Each such a c has degree $O(p^2)$ and size $O(p(M+\log p))$. Applying Proposition 1.3 (ii) we see that each run of such a RANI takes time

$$O\big(d^2(p^2+d\log d)(d(d+N)\log d + dp(M+\log p)+p^2N)^2\big)$$

and therefore we get the total time by multiplying this bound by p (since $O(p)$ is an upper bound for the number of c's).

ii) We perform $O(p)$ iterative steps. Each one of them computes at most $O(p)$ Sturm–Habicht sequences of polynomials of degrees $O(p)$ and $O(p\log p)$, carrying out in this way $O(p^3\log p)$ arithmetical operations with univariated polynomials.

These univariated polynomials have their sizes bounded by $O(p(p+M)\log p)$ and their degrees bounded by $O(p^2\log p)$. Thus, the computation of these sequences takes

$$O\big(p^9(\log p)^5(p+M)^2\big) \tag{1}$$

bit operations.

Once computed the Sturm–Habicht sequences, the RANI procedure must be called at most $O(p)$ times per queried Sth value. Each RANI has as input RAN(D) and a polynomial of degree $O(p^2\log p)$ and size $O(p(p+M)\log p)$. Thus, each such a RANI costs

$$O\big(d^2(p^2\log p + d\log d)(d(d+N)\log d + dp(p+M)\log p + p^2N\log p)^2\big).$$

It follows that, once the corresponding Sturm–Habicht sequences are known, the vectors of queries in each iterative step are computed in time

$$O\big(p^2d^2(p^2\log p + d\log d)(d(d+N)\log d + dp(p+M)\log p + p^2N\log p)^2\big). \tag{2}$$

Summing up (1) and (2) and multiplying the result by p (which is an upper bound for the number of iterative steps) we get the total time. ∎

2. The Algorithms.

Let $F \in \mathbb{Z}[X,Y]$ be a square–free polynomial, monic as polynomial in Y, and let D denote the discriminant of F with respect to Y. Let p be the degree of F and let $\xi_1 < \xi_2 < \ldots < \xi_r$ denote the real roots of D. Set $A_0 =]-\infty, \xi_1[$, $A_r =]\xi_r, +\infty[$ and, for $i = 1, \ldots, r-1$, $A_i =]\xi_i, \xi_{i+1}[$. (If $r = 0$, we shall write $A_0 = \mathbb{R}$.)

Let $C \subset \mathbb{R}^2$ be the algebraic set defined by the equation $F(X,Y) = 0$.

Let us recall that the number of real roots of $F(x,Y)$, considered as a polynomial in Y, remains constant as long as x lies in a (fixed) A_i. These roots are given by continuous semi–algebraic functions

$$\varphi_{i,1} < \varphi_{i,2} < \ldots < \varphi_{i,\rho_i} : A_i \longrightarrow \mathbb{R}, \quad i = 0, \ldots, r.$$

One calls the graphs of such functions $\varphi_{i,j}$ the ***real branches*** of C over A_i.

2.1. Cylindrical decomposition.

In this paragraph we give an algorithm CAD for the computation of a Cylindrical Decomposition of F (see [6], Sect. 2). Specifically, our algorithm

i) Characterizes the real roots ξ_i of D,

ii) Finds the number $\rho(\xi_i)$ of real points of $\mathcal{C}$ over each ξ_i, and

iii) Finds the number ρ_i of real branches of $\mathcal{C}$ over each A_i.

Notice that in particular this algorithm determines whether $\mathcal{C}$ is empty, a finite set of points, or an algebraic curve.

Algorithm CAD.

The algorithm CAD performs the following steps:

1) We compute the Sturm–Habicht sequence of F and F'_Y regarded as polynomials in Y. This computation gives D, up to a constant.

2) We compute the codes of the real roots ξ_i of D and η_j of D' ($i = 1, \ldots, r$, $j = 1, \ldots, s$). To do this, we apply RAN(D) and RAN(D') respectively.

Next, we sort all numbers ξ_i and η_j, by using their codes and Proposition 1.2 (ii). In particular, and because of Rolle's Theorem, for all i, $1 \leq i \leq r-1$, we find an index $j_i \in \{1, \ldots, s\}$ such that $\eta_{j_i} \in A_i$.

3) We compute the number $\rho(\xi_i)$ of real roots of $F(\xi_i, Y)$ and the number $\rho(\eta_j)$ of real roots of $F(\eta_j, Y)$ ($i = 1, \ldots, r$, $j = 1, \ldots, s$). To do this, we apply $\mathrm{St}_1(D, F)$ and $\mathrm{St}_1(D', F)$ respectively.

Notice that $\rho(\eta_{j_i}) = \rho_i$, for $i = 1, \ldots, r-1$. Moreover, if there is some j_0 (resp. j_r) such that $\eta_{j_0} < \xi_1$ (resp. $\eta_{j_r} > \xi_r$) then $\rho(\eta_{j_0}) = \rho_0$ (resp. $\rho(\eta_{j_r}) = \rho_r$).

4) If there does not exist such a root $\eta_{j_0} < \xi_1$ (resp. $\eta_{j_r} > \xi_r$), we find the number ρ_0 (resp. ρ_r) as the number of roots of $F(x, Y)$ when $x = -\infty$ (resp. when $x = +\infty$). To get this number, we only have to check the degrees and the signs of the leading coefficients of the Sturm–Habicht coefficients of F and F'_Y and to apply Theorem 1.1.

We have now the following result for the complexity of CAD.

Proposition 2.1.

Let n be the first integer greater than p and $|F|$. The algorithm CAD runs in time $O(n^{16}(\log n)^3)$.

Proof.

We only have to check the complexities of steps (1) to (4).

1) From Theorem 1.1 we deduce that the complexity of the computation of the Sturm–Habicht sequence of F and F'_Y is in $O(n^{10})$.

2) Both the degrees and the sizes of D and D' are bounded by $O(n^2)$. Then, by Proposition 1.3 (ii), the complexity of both RAN(D) and RAN(D') is in $O(n^{16}(\log n)^3)$.

3) By Proposition 1.4 (i), the complexity of both $\mathrm{St}_1(D, F)$ and $\mathrm{St}_1(D', F)$ is in $O(n^{15}(\log n)^3)$.

4) The complexity of this step is negligible. ∎

2.2. Topological type.

In this paragraph we give an algorithm TOP for the computation of a Semi–Algebraic Stratification of F (see [6] Sect. 2). Specifically, our algorithm

i) Characterizes the real roots ξ_i of D,

ii) Characterizes the real points $\zeta_{i,j}$ of $\mathcal{C}$ over each ξ_i,

iii) Characterizes the real branches $\varphi_{i,j}$ of $\mathcal{C}$ over each A_i, and

iv) Determines the adjacency relations between the points $\zeta_{i,j}$ and the branches $\varphi_{i',j'}$.

Notice that from this information we can deduce the topological type of $\mathcal{C}$: the number of connected components, the number of singularities, the number of algebraic branches through each singularity, etc ... We can even "draw" a planar graph homeomorphic to $\mathcal{C}$ (cf. [6]).

In order to get the information quoted in (i) and (ii), we can use RAN and RAN_1. For characterizing real branches of $\mathcal{C}$ in such a way that the adjacency relations between points and branches could be deduced, we shall rely upon the following facts, which were used for the first time, to our knowledge, in [6].

Proposition 2.2.

a) For each i, $1 \leq i \leq r$, there exists $\epsilon > 0$ such that for every $j \leq \rho_i$ the signs of $F'_Y, \ldots, F_Y^{p-1}$ remain constant over $\mathcal{C}_{i,j,+}$, the graph of $\varphi_{i,j}|_{]\xi_i,\xi_i+\epsilon[}$. It implies that the codes of the roots of $F(\nu, Y)$, for $\nu \in]\xi_i, \xi_i + \epsilon[$, characterize the real branches of $\mathcal{C}$ over A_i.

b) For $j \leq \rho_i$, if $\sigma = (\sigma_1 \ldots, \sigma_{p-1})$ is the $(p-1)$–tuple of sign conditions taken by $F'_Y, \ldots, F_Y^{(p-1)}$ over $\mathcal{C}_{i,j,+}$ then there exists one and only one root ζ of $F(\xi_i, Y)$ such that its code $\delta = (\delta_1, \ldots, \delta_{p-1})$ is compatible with σ. Moreover, $(\xi_i, \zeta) \in \overline{\mathcal{C}_{i,j,+}}$.

c) Similar claims hold on the left of ξ_i, $i = 1, \ldots, r$.

Proof.

a) Straightforward.

b) Let us consider ζ and ζ' roots of $F(\xi_i, Y)$, and let $\delta = (\delta_1, \ldots, \delta_{p-1})$ and $\delta' = (\delta'_1, \ldots, \delta'_{p-1})$ be their respective codes.

If both are compatible with σ, their $(p-1)$–tuples of relaxed sign conditions must coincide, and so, by Thom's Lemma, these two roots have to be the same.

On the other hand, at least one root of $F(\xi_i, Y)$ is adherent to $\mathcal{C}_{i,j,+}$ because F is monic as a polynomial in Y and therefore has no vertical asymptotes. A trivial argument of continuity entails the compatibility of this root's code with σ.

c) Just translate (a) and (b) to the left. ∎

Remark 2.1.

Point (b) of the previous Proposition was used by M.–F. Roy in [6] to decide to which points glued each real branch of $\mathcal{C}$. However, no real point was taken in $]\xi_i, \xi_i + \epsilon[$ in order to compute the $(p-1)$–tuples of sign conditions σ's corresponding to the branches over this interval.

Instead of that, a point ξ_i^+ "infinitesimally to the right" of ξ_i (and belonging to a non–archimedean extension of $\mathbb{R}$) was taken as a ground for such computations. These computations, that provide the part (iii) of the information quoted at the beggining of this paragraph, turns out to be the most expensive part of the algorithm given in [6], as it is shown in [8]. We shall return on these facts in §2.3.

Our idea is to decrease the cost of characterizing the real branches of $\mathcal{C}$ by working with real algebraic numbers instead of infinitesimals. Specifically, we shall find algebraic points $\eta_{i,+} \in]\xi_i, \xi_i + \epsilon[$, at least for $i = 1, \ldots, r-1$. To do so, first notice that if we denote by D_j the resultant w.r.t. Y of F and $F_Y^{(j)}$, $i \leq j \leq p-1$ (so that $D = D_1$), then the value $\xi_i + \epsilon$ can be taken as the nearest to ξ_i among all the roots at its right, of all D_j's. Let $\nu_{i,+}$ denote this root, and let j be such that $D_j(\nu_{i,+}) = 0$. Then, in the case $j \geq 2$ we can find a root $\eta_{i,+}$ of $(DD_j)'$ in $]\xi_i, \nu_{i,+}[$ (because of Rolle's Theorem), while in the case $j = 1$, $\eta_{i,+}$ can be found as a root of D'.

The only possible exception to this reasoning is for $i = r$, because such a $\nu_{r,+}$ does not have to exist. In this case, it turns out that the signs of $F'_Y, \ldots, F_Y^{p-1}$ remain constant over the graph of $\varphi_{r,j}|_{]\xi_r,+\infty[}$, $j = 1, \ldots, \rho_r$. So, in order to characterize the real branches of $\mathcal{C}$ over A_r we can take $\eta_{r,+} = +\infty$. This case will introduce no expensive computation, since the signs of univariated polynomials at $+\infty$ are trivially checked.

A similar reasoning applies to the left of ξ_i, at least for $i \geq 2$. ∎

The algorithm TOP.

Based on the latter remark, our algorithm TOP performs the following steps:

1) We compute the Sturm–Habicht sequence of F regarded as a polynomial in Y. As in the first step of CAD, this computation gives D, up to a constant.

We also compute $D_j = \text{resultant}\,(F, F_Y^{(j)})$, $j = 2, \ldots, p-1$.

2) We compute the codes of the real roots of D_j $(j = 1, \ldots, p-1)$, of $(DD_j)'$ $(j = 2, \ldots, p-1)$, and of D'. To do this, we apply $\text{RAN}(D_j)$ $(j = 1, \ldots, p-1)$, $\text{RAN}((DD_j)')$ $(j = 2, \ldots, p-1)$, and $\text{RAN}(D')$, respectively.

3) We apply $\text{RANI}(\text{RAN}(D_i), D_j^{(k)})$ $(1 \leq i < j \leq p-1,\ 0 \leq k \leq \deg D_j)$. By means of this information, applying Proposition 1.2 (ii), we sort the roots of all the D_j's.

We also perform RANI(RAN(D), $(DD_j)^{(k)}$) and RANI(RAN(D_j),$(DD_j)^{(k)}$) ($2 \leq j \leq p-1$, $0 \leq k \leq \deg DD_j$). By means of this information, and again by Proposition 1.2 (ii), we sort, for each $j \geq 2$, the roots of D, D_j and $(DD_j)'$.

Finally, we also sort the roots of D and D' by means of their codes.

In this way, for all couples (i,h) with $1 \leq i \leq r$ and $h \in \{+,-\}$ (except, possibly, for $(1,-)$ and $(r,+)$) we get an $i_h \in \{1,\ldots,p-1\}$, a root $\nu_{i,h}$ of D_{i_h} and, when $i_h > 1$, a root $\eta_{i,h}$ of $(DD_{i_h})'$, while if $i_h = 1$ then we get a root $\eta_{i,h}$ of D', in such a way that

i) $\nu_{i,-} < \eta_{i,-} < \xi_i$ and there is no root of any D_j in $]\nu_{i,-}, \xi_i[$

ii) $\xi_i < \eta_{i,+} < \nu_{i,+}$ and there is no root of any D_j in $]\xi_i, \nu_{i,+}[$.

4) We compute the codes of the real roots $\zeta_{i,j}$ of $F(\xi_i, Y)$, by means of $\mathrm{RAN}_1((\xi_i, D), F)$ ($1 \leq i \leq r$).

We also compute the codes of the real roots of $F(\eta_{i,h}, Y)$ ($1 \leq i \leq r$, $h \in \{-,+\}$). To do that, when $i_h = 1$ we apply $\mathrm{RAN}_1((D', \eta_{i,h}), F)$, and when $i_h > 1$ we apply $\mathrm{RAN}_1(((DD_{i_h})', \eta_{i,h}), F)$.

From all these codes, and Proposition 2.2, we can determine how the different real branches of $\mathcal{C}$ on A_k (at least for $k = 2, \ldots r-1$) glue to the $\zeta_{i,j}$'s. Moreover, if in step (3) we have found an element $\eta_{1,-}$ (resp. $\eta_{r,+}$) then we can also determine how the different real branches of $\mathcal{C}$ on A_0 (resp. A_r) glue to the $\zeta_{1,j}$'s (resp. $\zeta_{r,j}$'s).

5) If in step (3) we have not found such an element $\eta_{1,-}$ (resp. $\eta_{r,+}$), this implies that there does not exist either a real root $\nu_{1,-}$, of some D_{1-}, smaller than ξ_1 (resp. a real root $\nu_{1,+}$, of some D_{r+}, greater than ξ_r).

In this case, we compute the signs of $F'_Y, \ldots, F_Y^{(p-1)}$ on the roots of $F(-\infty, Y)$ (resp. $F(+\infty, Y)$), and we deduce from them how the different real branches of $\mathcal{C}$ on A_0 (resp. A_r) glue to the $\zeta_{1,j}$'s (resp. $\zeta_{r,j}$'s).

As far as the complexity of TOP goes, we have the following result:

Proposition 2.3.

Let n be the first integer greater than p and $|F|$. The algorithm TOP runs in $O(n^{19}(\log n)^3)$.

Proof.

We only have to check the complexities of steps (1) to (5).

1) The complexity of the computation of the Sturm–Habicht sequence of F and the complexity of the computation of D_j ($j = 2, \ldots, p-1$) are in $O(n^{10})$. So the total complexity of this step is in $O(n^{11})$.

2) We apply $O(n)$ times RAN to polynomials whose degrees and sizes are bounded by $O(n^2)$. By Proposition 1.3 (ii) the complexity of such an application of RAN is in $O(n^{16}(\log n)^3)$. Thus, the total complexity of this step is in $O(n^{17}(\log n)^3)$.

3) We apply $O(n^4)$ times RANI to couples of polynomials whose degrees and sizes are bounded by $O(n^2)$. By point (i) in Proposition 1.3, the complexity of such an application of RANI is in $O(n^{14}(\log n)^3)$. Thus, the total complexity of this step is in $O(n^{18}(\log n)^3)$.

4) We apply $O(n^2)$ times $\mathrm{RAN}_1((Q,\theta), F)$ to roots of polynomials Q whose degrees and sizes are bounded by $O(n^2)$. By Proposition 1.4 (ii) the complexity of such an application of $\mathrm{RAN}_1((Q,\theta), F)$ is in $O(n^{17}(\log n)^3)$. Thus, the total complexity of this step is in $O(n^{19}(\log n)^3)$.

5) The complexity of this step is negligible.

■

2.3 Comparison with Roy's algorithms.

As we have already said, in [6] an algorithm for the computation of the topological type of $\mathcal{C}$ is given, while an algorithm for the computation of a Cylindrical Decomposition of F can be found in [8]. Our algorithms are very close to these ones, except for the way of dealing with the real branches of $\mathcal{C}$ over the sets A_i: while we sistematically work with real algebraic numbers, in the quoted papers the authors use infinitesimals to cope whith what happens to the left or to the right of the roots ξ_i of D (see Remark 2.1). It turns out that our algorithms are less expensive than those in [6] and [8], and the goal of this paragraph is to explain why it is so.

Let us first recall some notations from [6] and [8].

Let $\xi \in \mathbb{R}$ and $Q \in \mathbb{Z}[X]$. One calls the ***sign of Q at ξ^+*** (resp. ***at ξ^-***) the sign of Q over an interval $]\xi, \xi+\epsilon[$ (resp. $]\xi-\epsilon, \xi[$), with $\epsilon > 0$ such that the sign of Q does not change on this interval.

Given $\xi \in \mathbb{R}$, one calls ***half-branch $\varphi_{\xi^+,j}$*** (resp. ***$\varphi_{\xi^-,j}$***) ***of $\mathcal{C}$ above ξ^+*** (resp. ***above ξ^-***) the germ of the graph of the function $\varphi_{i,j}$ on an interval $]\xi, \xi+\epsilon[$ (resp. $]\xi-\epsilon, \xi[$) where $\varphi_{i,j}$ is defined.

Given a polynomial $G \in \mathbb{Z}[X,Y]$, one calls ***the sign taken by*** G ***on*** $\varphi_{\xi^+,j}$ (resp. $\varphi_{\xi^-,j}$) the sign of $Q(x, \varphi_{i,j}(x))$, for $x \in]\xi, \xi + \epsilon'[$ (resp. $x \in]\xi - \epsilon', \xi[$), with $\epsilon' \leq \epsilon$ such that this sign is independent of x.

Remark 2.2.

We can determine the sign of a polynomial at ξ^+ (resp. at ξ^-) in the following way (see paragraph (I.C.2) in [8]):

i) If $Q(\xi) \neq 0$, then $\operatorname{sign} Q(\xi^+) = \operatorname{sign} Q(\xi^-) = \operatorname{sign} Q(\xi)$.

ii) If $Q(\xi) = 0$ and $Q^{(k)}$ is the first derivative of Q which does not vanish at ξ, then $\operatorname{sign} Q(\xi^+) = \operatorname{sign} Q^{(k)}(\xi)$ and $\operatorname{sign} Q(\xi^-) = (-1)^k \operatorname{sign} Q^{(k)}(\xi)$. ∎

Now, let $P, Q \in \mathbb{Z}[X]$, and let $\xi_1 < \ldots < \xi_r$ denote the real roots of P, which we suppose coded by RAN. The following algorithm evaluates the signs taken by Q at ξ_i^+ and ξ_i^-, $1 \leq i \leq r$.

Procedure RA $\mathbf{I}_1$**.**

Its ***input*** in the output of RAN(P) and the polynomial Q.

Its ***output*** is the list of signs Q takes at ξ_i^+ and ξ_i^-, $1 \leq i \leq r$.

The algorithm just applies RANI(RAN(P), $Q^{(k)}$), in increasing order of k, beginning with $k = 0$, and until for all ξ_i we have found a k_i such that $Q^{(k_i)}(\xi_i) \neq 0$. Then it applies the rules given in Remark 2.2.

An argument similar to the one used in the proof of Proposition 1.3 shows us that, with the notations of this Proposition, the running time of $\mathrm{RANI}_1(\mathrm{RAN}(P), Q)$ is bounded by

$$O\big(qd^2(q + d\log d)((d\log d + q)(d + N) + dS)^2\big).$$

Now, knowing the way of evaluating signs of polynomials at ξ^+ and ξ^-, one can give algorithms St_2 and RAN_2 for counting and coding respectively the half–branches of $\mathcal{C}$ over ξ^+ or ξ^-. These algorithms are similar to St_1 and RAN_1, except for the fact that they use RANI_1 instead of RANI. For the details, the reader can look up [8].

The bounds for the running times of these new algorithms can be obtained just by re–working the proof of Proposition 1.4, and using the complexity of RANI_1 instead of that of RANI. With the notations of that Proposition, the cost of $\mathrm{St}_2(D, P)$ turns out to be

$$O\big(p^3d^2(p^2 + d\log d)(d(d + N)\log d + dp(M + p) + p^2N)^2\big)$$

and the cost of $\mathrm{RAN}_2((P, \xi^+), F)$ (or $\mathrm{RAN}_2((P, \xi^-), F)$) is

$$O\big(p^{10}(\log p)^5(p + M)^2 + p^5d^2\log p(p^2\log p + d\log d)(d(d + N)\log d + dp(p + M)\log p + p^2N\log p)^2\big)$$

Of course, these running times are much bigger than those for St_1 and RAN_1, since the cost of RANI_1 is bigger than that of RANI. It shows us that it is more expensive to work with infinitesimals of the form ξ^+ or ξ^- than to work with algebraic numbers.

Let us come back now to Roy's algorithms. Let us consider first the algorithm CD in [8], which gives a Cylindrical Decomposition of F. The difference between this algorithm and our algorithm CAD is in the way of getting point (iii) of the information quoted at the beginning of §2.1. While CAD computes the number of real branches over each A_i as the number of roots of $F(\eta, Y)$, with η a suitable real algebraic number whose code we know (except possibly for $i = 0, r$, in which case the extra cost is negligible), CD computes this number of real branches as the number of half–branches of $\mathcal{C}$ above ξ^+ (except for $i = 0$, in which case it takes ξ_1^-). So, we use St_1, while M.–F. Roy uses St_2, which is more expensive. In fact, it is easy to check that the dominant step of CD is $\mathrm{St}_2(D, F)$, which has a complexity in $O(n^{17}(\log n)^3)$.

Let us consider now the algorithm TOP in [6], to which we shall refer in the sequel as TOPR, that computes the topological type of $\mathcal{C}$. The difference between this algorithm and our algorithm TOP is in the way of getting points (iii) and (iv) of the information quoted at the beginning of §2.2. While TOP characterizes the real branches to the left and to the right of each ξ_i by coding the roots of $F(\eta, Y)$, with η a suitable real algebraic number with known code (except possibly for the real branches to the left of ξ_1 and to the right of ξ_r, in which case the extra cost is negligible), TOPR characterizes them by coding the half–branches over ξ_i^- and ξ_i^+. So we use RAN_1, while M.–F. Roy uses RAN_2, which is again much more expensive. In fact, it is not difficult to show that the dominant step of TOPR is $\mathrm{RAN}_2(D, F)$, whose complexity is in $O(n^{21}(\log n)^4)$.

References.

[1] D. Arnon and S. Mac Allum; "A polynomial-time algorithm for the topological type of a real algebraic curve". *J. of Symb. Comp.*, **5**, 1988.

[2] M. Coste and M.-F. Roy; "Thom's lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets". *J. of Symb. Comp.* **5**, 1988.

[3] F. Cucker and M.-F. Roy; "A theorem on random polynomials and some consequences in average complexity", to appear in *J. of Symb. Comp.*

[4] P. Gianni and C. Traverso; "Shape determination of real curves and surfaces". *Ann. Univ. Ferrara, Sez. VII, Sec. Math.*, vol **XXIX**, 1983.

[5] L. González, H. Lombardi, T. Recio and M.-F. Roy; "Spécialisation de la suite de Sturm et sous-résultants", to appear in *RAIRO Inf. Théor. et Appl.*

[6] M.-F. Roy; "Computation of the topology of a real algebraic curve", to appear in the *Proceedings of the Conference on Computational geometry and topology*, Sevilla, 1987.

[7] M.-F. Roy and A. Szpirglas; "Complexity of computations on real algebraic numbers", to appear in *J. of Symb. Comp.*

[8] M.-F. Roy and A. Szpirglas; "Complexity of computations of cylindrical decomposition and topology of real algebraic curves using Thom's lemma", to appear in the *Proceedings of the Conference on Real Algebraic Geometry*, Trento, 1988.