
Improved Algorithms for Sign Determination and Existential Quantifier Elimination

JOHN CANNY*

Computer Science Division, University of California, Berkeley 94720

Recently there has been a lot of activity in algorithms that work over real closed fields, and that perform such calculations as quantifier elimination or computing connected components of semi-algebraic sets. A cornerstone of this work is a symbolic sign determination algorithm due to Ben-Or, Kozen and Reif [2]. In this paper we describe a new sign determination method based on the earlier algorithm, but with two advantages: (i) It is faster in the univariate case, and (ii) In the general case, it allows purely symbolic quantifier elimination in pseudo-polynomial time. By purely symbolic, we mean that it is possible to eliminate a quantified variable from a system of polynomials no matter what the coefficient values are. The previous methods required the coefficients to be themselves polynomials in other variables. Our new method allows transcendental functions or derivatives to appear in the coefficients.

Another corollary of the new sign-determination algorithm is a very simple, practical algorithm for deciding existentially-quantified formulae of the theory of the reals. We present an algorithm that has a bit complexity of $n^{(k+1)}d^{O(k)}(c \log n)^{(1+\epsilon)}$ randomized, or $n^{(k+1)}d^{O(k^2)}c^{(1+\epsilon)}$ deterministic, for any $\epsilon > 0$. Where n is the number of polynomial constraints in the defining formula, k is the number of variables, d is a bound on the degree, c bounds the bit length of the coefficients. The algorithm makes no general position assumptions, and its constants are much smaller than other recent quantifier elimination methods.

Received November 1992, revised April 1993

1. INTRODUCTION

Real quantifier elimination algorithms have enjoyed a revival in interest in the last few years, and an important impetus was the parallel sign determination algorithm in [2]. Two of the latest quantifier elimination algorithms, [21] and [14] have very good theoretical complexity. Indeed the bounds in [21] are near the best that can be hoped for the important case of problems with a bounded number of quantifier alternations. Their work builds on a large body of earlier papers, dating from Tarski's original paper [24], and following a line of papers that include [23], [9], [11], [10], [2], [3] and others.

Both [21] and [14], like the author's earlier paper [4], make use of a sign-determination lemma due to Ben-Or, Kozen and Reif [2]. This lemma, henceforth called "BKR", takes a univariate polynomial $p(s)$, and polynomials $q_1(s), \dots, q_n(s)$, and returns k sign sequences $\sigma \in \{-, 0, +\}^n$, where k is the number of real roots of $p(s) = 0$. Each sign sequence σ corresponds to a particular root α of $p(s) = 0$, in such a way that $\sigma_i = \text{sign}(q_i(\alpha))$. Indeed BKR is an indispensable component of these works, providing two benefits: (i) The

ability to work over any real closed field, (ii) The possibility of efficient (NC) parallel implementation. In [18] an ingenious parallel algorithm was given for approximating all the roots of a univariate polynomial. His algorithm can easily be used to parallelize earlier algorithms, like [9], so that (ii) can now be achieved without BKR. But the benefits of being able to work over arbitrary real closed fields should not be underestimated, and BKR remains the most effective tool for this. For example, infinitesimal extensions of the reals, which are themselves real closed fields, are extensively used in recent algorithms for computing connected components of semi-algebraic sets [3], [15], [11], [5], [6].

BKR also seems to perform well in implementations, especially if the polynomial $p(s)$ has few real roots. Although definitive testing has not been done as yet, in our recent implementations, it is often faster to determine the signs of polynomials symbolically using BKR than to compute roots of $p(s) = 0$ numerically and then substitute to find the signs of the $q_i(s)$'s. Since any comparison can be reduced to a sign test of a single polynomial, sign determination is a "universal" calculation, or at least as general as root-finding and substitution.

Given the importance of BKR, it is natural to look for improvements and simplifications. This paper is motivated in part by a desire to improve BKR, and in part

*Supported by a David and Lucile Packard Foundation Fellowship and by NSF Presidential Young Investigator Grant IRI-8958577.

by a feeling that a more uniform algorithm (fewer potential branches in the calculation) should be possible. For the most part, these goals have been realised. In BKR, the number of potential branches is exponential. More precisely, BKR makes all its decisions based on sign queries of various polynomials that it computes along the way in the coefficients of $p(s)$ and the $q_i(s)$'s. The number of potential queries in BKR is exponential.

In this paper, we present a sign determination algorithm which has only a pseudo-polynomial number of potential queries. This makes possible a very simple recursive algorithm for sign determination in the multivariate case, where $p(s)$ and $q_i(s)$ have coefficients which depend on other variables. We simply compute all the potential query polynomials, and return a formula that depends only on their signs. This cannot be done efficiently with BKR, and [14] and [21] use clever subterfuges. Both papers sacrifice generality for efficiency, however. They work only if the coefficients of $p(s)$ and $q(s)$ are themselves polynomials in other variables. Our new method does not have this limitation. It works for any $p(s)$ and $q_i(s)$, without restriction on the coefficients, e.g. they may be exponential functions or derivatives of other variables.

Our algorithm is also simple to implement. Like BKR, it works by solving linear systems $Ax = b$ to find the sign combinations. Whereas BKR recursively computes the matrix A by tensoring, our new algorithm allows A to be computed directly. We also get better bounds on the degree of the query polynomials. In this paper, multivariate sign determination follows the method of [4]. An entirely different approach to sign determination is the multivariate Sturm sequence technique is presented in [19] and [20].

We then describe a simple, practical algorithm for deciding existentially-quantified formulae of the theory of the reals. We present a randomized algorithm for that with a bit complexity of $n^{(k+1)}d^{O(k)}(c \log n)^{(1+\epsilon)}$ randomized, or $n^{(k+1)}d^{O(k^2)}c^{(1+\epsilon)}$ deterministic, for any $\epsilon > 0$. This takes as input a formula with n polynomial constraints in k variables of degree at most d , and c -bit coefficients. The algorithm makes no general position assumptions, and its constants are much smaller than other recent quantifier elimination methods. An implementation is underway, and the results will be reported soon.

2. SIGN DETERMINATION

If $f(s)$ and $g(s)$ are polynomials, let the Sturm sequence of f and g be denoted r_0, r_1, \dots, r_k , where $r_0(s) = f(s)$, $r_1(s) = g(s)$, and the intermediate remainders are computed via

$$r_{i+1} = q_i r_i - r_{i-1} \quad (1)$$

where q_i is the quotient of the polynomial division of r_{i-1} by r_i . In practice, pseudo-remainders are often used so that the r_i have integer coefficients. For a real

value v , let $\text{SA}(f, g, v)$ denote the number of changes in sign in the sequence $r_0(v), r_1(v), \dots, r_k(v)$.

Let $\text{SC}(f, g)$ denote the quantity $\text{SA}(f, g, +\infty) - \text{SA}(f, g, -\infty)$. The classical Sturm theorem states that $\text{SC}(f, f')$ equals the number of real roots of $f(s) = 0$. The most general form of the theorem states that $\text{SC}(f, f'g)$ gives the sum of the number of real roots of $f(s) = 0$ where $g(s) > 0$, minus the number where $g(s) < 0$, common roots making no difference to the count. A useful application of this result is to consider $\text{SC}(f, f'g^2)$ which counts real roots of $f = 0$ where $g \neq 0$. This observation was used by [22] to simplify the BKR sign determination lemma.

LEMMA 2.1. [22] *Let r^+ denote the set of roots of $f(s) = 0$ where $g(s)$ is positive, and similarly for r^0 and r^- . Then the numbers of roots in each set satisfy the following identity*

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} |r^+| \\ |r^-| \\ |r^0| \end{bmatrix} = \begin{bmatrix} \text{SC}(f, f') \\ \text{SC}(f, f'g) \\ \text{SC}(f, f'g^2) \end{bmatrix} \quad (2)$$

Now we can apply this lemma to find the signs of a system of polynomials $q_1(s), \dots, q_n(s)$ at the roots of a single polynomial $p(s) = 0$. We start by applying the above lemma to $p(s)$ and $q_1(s)$. We can rewrite the above identity in matrix form $H_1 R_1 = S_1$, where H_1 is the 3×3 matrix, R_1 is the vector of root counts for q_1 and S_1 is the vector of Sturm query counts for q_1 :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} |r_1^+| \\ |r_1^-| \\ |r_1^0| \end{bmatrix} = \begin{bmatrix} \text{SC}(p, p') \\ \text{SC}(p, p'q_1) \\ \text{SC}(p, p'q_1^2) \end{bmatrix} \quad (3)$$

There is a more general identity which we can write as $H_n R_n = S_n$ where we define H_n inductively as

$$H_{n+1} = \begin{bmatrix} H_n & H_n & H_n \\ H_n & -H_n & 0 \\ H_n & H_n & 0 \end{bmatrix} \quad (4)$$

and where R_{n+1} is the 3^{n+1} vector obtained by listing the elements of R_n with r_{n+1}^+ appended to each conjunction, then the elements of R_n with r_{n+1}^- appended, and finally a last copy of R_n with r_{n+1}^0 . Similarly, S_{n+1} is obtained by listing first a copy of S_n , then another copy of S_n with q_{n+1} multiplying the derivative in each Sturm sequence, and then a third copy of S_n with q_{n+1}^2 . We can write these definitions schematically as

$$R_{n+1} = \begin{bmatrix} R_n \cap r_{n+1}^+ \\ R_n \cap r_{n+1}^- \\ R_n \cap r_{n+1}^0 \end{bmatrix} \quad S_{n+1} = \begin{bmatrix} S_n \\ S_n \cdot q_{n+1} \\ S_n \cdot q_{n+1}^2 \end{bmatrix} \quad (5)$$

and it is easy to verify that $H_{n+1} R_{n+1} = S_{n+1}$. It is also easy to see that H_{n+1} is non-singular. Simply add the middle rows to the last rows to give a block diagonal matrix whose determinant is non-zero if the determinant of H_n is.

Of course, the problem with doing things this way is that we need to invert a $3^n \times 3^n$ matrix to compute

the signs of n univariate polynomials. One of the major contributions of [2] is a clever method to avoid all this calculation. They observed that if $p(s) = 0$ has k real roots, then at most k of the elements of R_n can be non-zero, since the elements of R_n form a partition of the roots of $p(s) = 0$.

Suppose then, that one has calculated the at most m sign sequences of q_1, \dots, q_n , which will be counted in $\leq m$ non-zero elements of R_n . When we come to compute R_{n+1} we can use the fact that there are three possible signs for q_{n+1} at each root, hence only $\leq 3m$ possible non-zero elements in R_{n+1} . For the other elements we can immediately fill in zeros, and in fact we can delete the corresponding columns of H_{n+1} , giving a $3^n \times 3m$ submatrix. To solve the system, we need only find $3m$ independent rows of this submatrix, and solve a $3m \times 3m$ system. Finding these rows might potentially take a long time, but [2] show that one can use m independent rows from R_n from the previous step, so the whole calculation takes polynomial time (actually, the description above is for a simplified sequential version of their algorithm, the original [2] uses divide-and-conquer to run in NC).

This is fine for univariate problems, since one never needs to make more than $3mn$ Sturm queries. But it is not known in advance which queries will be made, so the number of *potential* queries is still 3^n . This causes problems when one tries to use the algorithm symbolically. This wasn't fully appreciated in [2], and while their univariate analysis is correct, the multivariate generalization given there is not. The present work was originally motivated by an attempt to correct this error in the simplest way possible, namely by finding good a priori bounds on the number of potential Sturm queries.

It seems that this should be possible for the following reason: The number of potential Sturm queries corresponds to the number of rows of H_n that might ever be used during the algorithm. Since the basic operation is to find $3m$ independent rows given $3m$ columns, a natural step is to find a subset of rows of H_n with the property that any $3m$ columns are linearly independent. Then whatever columns we need to work with, we can still be guaranteed to find independent rows within our new submatrix.

Ideally, this submatrix would have a polynomial number of rows. We will not succeed in this, but we do not miss by much. The number of rows turns out to be pseudo-polynomial in m . This is enough to give us some interesting new bounds for more general quantifier elimination.

DEFINITION 2.1. *Our new matrix is denoted $K_{n,m}$ which indicates that it has 3^n columns and any m of them are linearly independent. Thus it is defined only if $m \leq 3^n$. If $m = 1$, $K_{n,m}$ is just the first row of H_n , i.e. a row of all ones. If $m > 3^{n-1}$, then $K_{n,m}$ is just*

H_n . Otherwise, it is defined recursively as follows:

$$K_{n,m} = \begin{bmatrix} K_{(n-1),m} & K_{(n-1),m} & K_{(n-1),m} \\ K_{(n-1),\lfloor \frac{m}{2} \rfloor} & -K_{(n-1),\lfloor \frac{m}{2} \rfloor} & 0 \\ K_{(n-1),\lfloor \frac{m}{3} \rfloor} & K_{(n-1),\lfloor \frac{m}{3} \rfloor} & 0 \end{bmatrix} \tag{6}$$

Now we must show that this definition does indeed give us m independent columns whichever ones we choose.

LEMMA 2.2. *Any m columns of $K_{n,m}$ as defined above are linearly independent.*

Proof. For $m > 3^{n-1}$ the result is immediate, since H_n is non-singular. So we assume it is true for all values of n less than or equal to our given n , and for all values of m less than or equal to the given m .

Now consider the definition (6), and pick any set of m columns. We suppose that there exists a linear combination of these columns which is zero, and show that all the coefficients are zero. We do this by considering "top sections" of the columns, which are the rows in the first group of rows in (6). Specifically, let c_{i_1}, \dots, c_{i_m} be the columns, and suppose that

$$\sum_{j=1}^m \lambda_{i_j} c_{i_j} = 0 \tag{7}$$

Now consider just the top sections of those columns, whose rows lie in $K_{(n-1),m}$. Let c'_{i_j} denote the top section of c_{i_j} . Since up to three columns may have the same top section, the number q of distinct c' 's is between $\frac{m}{3}$ and m , depending on how many columns match in top sections. The linear combination (7) of columns from the previous paragraph gives us

$$\sum \lambda_{i_j} c'_{i_j} = 0 \tag{8}$$

But the c' 's are just columns of $K_{(n-1),m}$ and any m columns of $K_{(n-1),m}$, which appears in the top section of $K_{n,m}$, are linearly independent. So any column section that appears once in (8) must have its $\lambda = 0$.

For the other λ 's, let us define for $j = 0, 1, 2, 3$ a set of column indices $C(j)$ as

$$C(j) = \{i \in [0, \dots, 3^{(n-1)}] \mid \text{precisely } j \text{ elements of } \lambda_i, \lambda_{i+3^{(n-1)}}, \lambda_{i+2 \cdot 3^{(n-1)}} \text{ are non-zero}\}$$

and by the argument just made, $C(1)$ is empty.

Now consider $C(2)$. For each $i \in C(2)$, the columns $c_i, c_{i+3^{(n-1)}}$ and $c_{i+2 \cdot 3^{(n-1)}}$ all have the same top section, c'_i , and its coefficient in (8) must be zero. So we know

$$(\lambda_i + \lambda_{i+3^{(n-1)}} + \lambda_{i+2 \cdot 3^{(n-1)}}) = 0 \tag{9}$$

where exactly two of these λ s are non-zero.

But by considering the middle sections of those columns, from (6) and (7), we see that

$$(\lambda_i - \lambda_{i+3^{(n-1)}}) = 0 \tag{10}$$

This identity holds because there are at most $\lfloor \frac{m}{2} \rfloor$ elements in $C(2)$, and any $\lfloor \frac{m}{2} \rfloor$ elements of $K_{(n-1),\lfloor \frac{m}{2} \rfloor}$,

which appears in the middle section of $K_{n,m}$, are linearly independent. But equations (9) and (10) imply that the three λ s are scalar multiples of each other. Since one of them is zero, so are the other two. So $C(2)$ is in fact empty.

Now consider $C(3)$. Any $i \in C(3)$ implies that both the identities (9) and (10) hold, from which it follows that

$$2\lambda_i = 2\lambda_{i+3(n-1)} = -\lambda_{i+2*3(n-1)}$$

This time there are at most $\lfloor \frac{m}{3} \rfloor$ elements in $C(3)$. Since any $\lfloor \frac{m}{3} \rfloor$ elements of $K_{(n-1), \lfloor \frac{m}{3} \rfloor}$ are linearly independent, considering bottom sections of the columns implies the identity

$$\lambda_i + \lambda_{i+3(n-1)} = 0$$

So the only solution is again $\lambda_i = \lambda_{i+3(n-1)} = \lambda_{i+2*3(n-1)} = 0$. We conclude that $C(3)$ is empty, also. Since all λ 's in (7) must be zero, we conclude any m of the c_{i_j} are linearly independent. ■

Observation $K_{n,m}$ is a submatrix of H_n , consisting of rows of H_n .

This observation is easily proved by induction using the recursive definitions of H_n and $K_{n,m}$ in equations (4) and (6).

Next we make a simple observation which provides good bounds on the degrees of the polynomials in the Sturm queries. Since $K_{n,m}$ is a submatrix consisting of rows of H_n , the product $K_{n,m}R_n$ is a subvector of S_n . Let ${}_{,n,m}$ be this subvector, which consists of rows of S_n which correspond to the rows of $K_{n,m}$. Then

$${}_{,n,m} = K_{n,m}R_n \tag{11}$$

and we observe the following:

LEMMA 2.3. *The maximum number of q_i 's occurring in any Sturm query in ${}_{,n,m}$ is $\lfloor \log_2(m) \rfloor$.*

Proof. The proof is by induction on n and m . Now ${}_{,n,m}$ can be recursively defined analogously to (5) using (6) as:

$${}_{,n+1,m} = \begin{bmatrix} {}_{,n,m} \\ {}_{,n, \lfloor \frac{m}{2} \rfloor} \cdot q_{n+1} \\ {}_{,n, \lfloor \frac{m}{3} \rfloor} \cdot q_{n+1}^2 \end{bmatrix} \tag{12}$$

and the recursion leads eventually to one of the two base cases (i) if $m = 1$, then ${}_{,n,m} = S_0$, or (ii) the case $m > 3^{n-1}$, when ${}_{,n,m} = S_n$. We prove the bound for both of these cases, and then by induction for the intermediate ${}_{,n,m}$.

For the first base case, $m = 1$, none of the q_i 's appear in a Sturm query, and the bound holds. For the second case, since ${}_{,n,m} = S_n$, n is exactly the maximum number of q_i 's occurring in Sturm queries. Now n is less than $\log_3(m) + 1$, and since it is integer, it is

bounded by $\lfloor \log_3(m) + 1 \rfloor$, which is less than or equal to $\lfloor \log_2(m) \rfloor$ for $m \geq 2$. Since $m = 1$ was already dealt with, this shows that the bound holds for all base cases.

Now for the inductive step, we have only to inspect (12) to notice that if the inductive hypothesis holds for all cases on the right-hand side of (12), then it holds for the left-hand side. The polynomial q_{n+1} is only added to Sturm query vectors ${}_{,n, \lfloor \frac{m}{2} \rfloor}$ and ${}_{,n, \lfloor \frac{m}{3} \rfloor}$ which have at most $\lfloor \log_2(m) - 1 \rfloor$ polynomials, by hypothesis. Since the recursion (12) produces a finite tree of intermediate $K_{n,m}$'s with leaves which are base cases, and since the bound holds for a parent whenever it holds for the descendants, this completes the proof. ■

COROLLARY 2.4. *The number of rows in ${}_{,n,m}$ or $K_{n,m}$ is*

$$\binom{n}{\lfloor \log_2(m) + 1 \rfloor} 2^{\lfloor \log_2(m) \rfloor} = (n^{O(\log m)}) \tag{13}$$

Proof. Each entry in ${}_{,n,m}$ is a Sturm query, and can be uniquely indexed by (i) the set of (at most $\log(m)$) q_i 's that occur in it and (ii) the vector of exponents of these q_i 's which are all either 1 or 2. The bound comes from first counting the number of subsets of at most $\log(m)$ q_i 's, which is $\binom{n}{\lfloor \log_2(m) + 1 \rfloor}$ and then counting the number of possible exponent vectors for each, which is $2^{\lfloor \log_2(m) \rfloor}$. ■

3. AN ALGORITHM FOR DETERMINING THE SIGN

In this section we show how to turn the results of the previous section into a sign-determination algorithm. Although there are similarities with BKR, there is a major difference. Instead of constructing a submatrix of H by tensoring submatrices from the previous step, our method computes a submatrix of K directly from the set of possible sign sequences. The algorithm works inductively as follows (d is the degree of $p(s)$):

- We assume at the i^{th} step that the algorithm knows for each sign sequence of q_1, \dots, q_{i-1} , how many roots of $p(s) = 0$ produce this sign sequence. Most of these are zero, and the algorithm only stores the sign sequences with at least one root, and the number of these is $m_{i-1} \leq d$.
- There are $3m_{i-1}$ possible sign sequences for q_1, \dots, q_i , and each of these defines a column of the matrix $K_{i,3m_{i-1}}$. These columns are linearly independent, so there are $3m_{i-1}$ rows which together with the specified columns, define a square submatrix J_i of $K_{i,3m_{i-1}}$.
- We solve the $3m_{i-1} \times 3m_{i-1}$ system corresponding to this matrix, to find the actual sign sequences of q_1, \dots, q_i , and repeat the above steps for $i = 1, \dots, n$.

The first task then, is to give a procedure that accepts a list of m columns of the matrix $K_{n,m}$, and returns a list of m rows, such that the resulting $m \times m$ matrix is

non-singular. This procedure needs to run in polynomial time in n and m and not the size of $K_{n,m}$.

The second task, which is very easy, is to determine the entries of this submatrix of $K_{n,m}$. Again this must be in polynomial time in m , so we cannot afford to construct all of $K_{n,m}$. This task reduces to determining the value of a single element of $K_{n,m}$ given its row and column indices.

3.1. Computing rows given columns of K

To simplify our representation of K , we use the fact that $K_{n,m}$ is a submatrix of H_n , consisting of rows of H_n . To reference a row of $K_{n,m}$, we use the row index of the corresponding row of H_n . So $(K_{n,m})_{i,j} = (H_n)_{i,j}$. In this representation, $K_{n,m}$ has many missing rows, but the row indices are always produced by the algorithm we are about to describe, and we ensure that it always returns a row index that does lie in $K_{n,m}$.

Each column index is a trinary number that corresponds in an obvious way with a particular sign sequence. The sign sequence $\text{sign}(q_1), \dots, \text{sign}(q_n)$ encodes as the trinary number $t_n t_{n-1} \dots t_1$, where

$$t_i = \begin{cases} 0 & \text{if } \text{sign}(q_i) = + \\ 1 & \text{if } \text{sign}(q_i) = - \\ 2 & \text{if } \text{sign}(q_i) = 0 \end{cases} \quad (14)$$

Let the trinary column indices corresponding to the m sign sequences be c_1, \dots, c_m . The first observation we make is that we don't need to pass the algorithm the value of n . Each c_i is an n -digit trinary number, but it is possible that all the leading digits are zero, so they look like $(n-1)$ -digit or shorter numbers. If this happens, it is perfectly OK to treat them as shorter numbers, because the corresponding set of rows would come from the matrix $K_{(n-1),m}$, even if we had treated them as n -trit numbers. A close inspection of the results of the previous section will confirm this.

Algorithm Rows(L)

The input $L = (c_1, \dots, c_m)$ is a list of m column indices, and the output is a list of m row indices.

- If $m = 0$ return the empty list. If $m = 1$, return a list of zero (the index of the first row).
- Otherwise, we determine n by computing $n = 1 + \max(\lfloor \log_3(c_i) \rfloor)$, $i = 1, \dots, m$, which is the maximum number of digits in any column index.
- We compute from L , three lists L_1, L_2 and L_3 whose total length is m . These lists contain elements of L reduced modulo 3^{n-1} . The set L_1 consists of all values $c_i \bmod 3^{n-1}$, whereas L_2 and L_3 consist of elements which are duplicated or occur three times.

Specifically, we have

$$\begin{aligned} L_1 &= \{k \mid k = c_i \bmod 3^{n-1} \text{ for some } i\} \\ L_2 &= \{k \mid k = c_i \bmod 3^{n-1} \text{ for } \geq 2 \text{ values of } i\} \\ L_3 &= \{k \mid k = c_i \bmod 3^{n-1} \text{ for 3 values of } i\} \end{aligned} \quad (15)$$

- We then call our algorithm recursively on each non-empty list, and return the list:

$$\text{Rows}(L_1) \circ (3^{n-1} + \text{Rows}(L_2)) \circ (2 * 3^{n-1} + \text{Rows}(L_3)) \quad (16)$$

where \circ denotes concatenation of lists, and where the plus sign between a number and a list means that the number is to be added to every element of the list. The two additions effectively shift the rows returned by $\text{Rows}(L_2)$ and $\text{Rows}(L_3)$ to the middle and bottom bands of $K_{n,m}$ respectively.

The proof of correctness for the algorithm mimics the lemmas of the previous section very closely, and we do not repeat it here. The algorithm is easily seen to be polynomial time, because the recursion depth is at most n , and the total number of columns in *all* calls at any level is m , so the total number of calls is not more than nm .

3.2. Computing elements of K

To compute the elements of the submatrix of $K_{n,m}$ consisting of a particular set of m rows and columns, we need only show that there is an efficient method for computing the element $(K_{n,m})_{r,c}$ given the row and column indices r and c . Since we chose the row indices to correspond to elements of H_n , this element is the same as $(H_n)_{r,c}$. Finally, note that we do not even need to know the value of n explicitly, because $(H_n)_{r,c} = (H_{n-1})_{r,c}$ if r and c are both less than 3^{n-1} .

Algorithm Helt(r,c)

The input is a pair of trinary integers r and c , and the output is the corresponding element of H_n , which is in $\{-1, 0, 1\}$.

- Determine $n = 1 + \max(\lfloor \log_3(r) \rfloor, \lfloor \log_3(c) \rfloor)$, the size of the smallest H_n containing this element.
- Let the trinary expansions of r and c be $r_n r_{n-1} \dots r_1$ and $c_n c_{n-1} \dots c_1$ respectively. Compute and return the value of the product

$$\prod_{i=1}^n (H_1)_{r_i, c_i} \quad (17)$$

where H_1 is the 3×3 matrix defined earlier.

This expression for the value of $(H_n)_{r,c}$ is obtained by unrolling a recursive algorithm based on the definition in (4).

3.3. Complexity Analysis

Let m denote the number of real roots of $p(s) = 0$, then m is not more than d , the degree of $p(s)$. The algorithm takes n steps to compute the signs of all q_i 's and each step involves inverting a $3m \times 3m$ matrix and the calculation of $3m$ Sturm queries. Each Sturm query involves $p(s)$ and a product of at most $2 \log_2(m)$ of the q_i 's. The number of arithmetic operations for each Sturm query is $O(d \log m \log^2(d \log m))$, which simplifies to $O(d \log m \log^2 d)$ because d is at least as large as m . So the total time per step for the Sturm calculation is

$$O(md \log m \log^2 d) \tag{18}$$

or $O(d^2 \log^3 d)$ in the worst case when the number of roots is roughly d . The cost of inverting the matrix is $O(m^{2.376})$ and so the overall running time is:

$$O(n(md \log m \log^2 d + m^{2.376})) \tag{19}$$

The cost of inverting the matrix dominates if m is close to d . However, the actual number of real roots is often much less than d , and is $O(\log d)$ for a random polynomial. So the Sturm query time will often be the larger in practice, especially if naive algorithms are used. Compared to BKR in the univariate case, we have removed a factor of at least n , since the Sturm queries in BKR may contain polynomials of degree dn . This makes possible the very simple existential quantifier elimination algorithm of the next section.

4. A DECISION ALGORITHM FOR EXISTENTIAL FORMULAE

The input to the algorithm is a formula of the form $B(A_1, \dots, A_n)$ where $B : \{0, 1\}^n \rightarrow \{0, 1\}$ is a boolean function and each A_i is an atomic formula of one of the following types:

$$(f_i = 0), (f_i \neq 0), (f_i > 0), (f_i < 0), (f_i \geq 0), (f_i \leq 0) \tag{20}$$

with each f_i a polynomial in x_1, \dots, x_k with rational (for our computational purposes) coefficients. In the method that follows it will be helpful to assume (wlog) a certain form for the defining predicate:

DEFINITION 4.1. *A formula $B(A_i, \dots, A_n)$ is said to be in monotone standard form if the boolean function B is monotone, and all atomic formulae A_i are either $(f_i = 0)$ or $(f_i > 0)$.*

As shown in [6] we can go further and assume that the formula has log depth. But that will not be of advantage to us for this simple algorithm.

Predicate complexity We measure the complexity of a predicate with four quantities, the number of polynomials n , the number of variables k , the maximum degree of the polynomials d , and the maximum coefficient length c of the coefficients of the polynomials.

4.1. Stratifications

DEFINITION 4.2. *A stratification \underline{S} of a set $S \subset \mathbb{R}^k$ is a partition of S into a finite number of disjoint subsets S_i called strata such that each S_i is a manifold.*

A regular stratification satisfies some additional conditions which are well described in [13]. There are several ways to construct regular stratifications. We will only need two:

- **Taking products.** If \underline{C} and \underline{D} are regular stratifications of the spaces C and D respectively, then the product $\underline{C} \times \underline{D}$ is a regular stratification of $C \times D$.
- **Preimage of a transversal map.** If $F : M \rightarrow N$ is transversal to \underline{D} for a regular stratification \underline{D} of a subset $D \subseteq N$, then $F^{-1}(\underline{D})$ is a regular stratification of $F^{-1}(D)$.

where $F \bar{\cap} \underline{D}$ means F is transversal to all the strata of \underline{D} , and $F^{-1}(\underline{D})$ is the set $\{F^{-1}(\sigma) \mid \sigma \in \underline{D}\}$. If we now define

DEFINITION 4.3. *Let $f_i \in \mathbb{Q}[x]$, $i = 1, \dots, n$ be a collection of polynomials that define a map $F = (f_1, \dots, f_n) : \mathbb{R}^k \rightarrow \mathbb{R}^n$. Let $\underline{\mathbb{R}} = \{\mathbb{R}_-, \{0\}, \mathbb{R}_+\}$, a sign sequence σ is an element of $(\underline{\mathbb{R}})^n$. The sets $F^{-1}(\sigma)$ are called sign-invariant sets of F .*

Then we can view a semi-algebraic set S as a finite union of sign-invariant sets of some polynomial map F . The sign partition $(\underline{\mathbb{R}})^n$ of \mathbb{R}^n is a regular stratification of \mathbb{R}^n . So if a map $F : \mathbb{R}^k \rightarrow \mathbb{R}^n$ is transversal to $(\underline{\mathbb{R}})^n$, then the preimage $F^{-1}((\underline{\mathbb{R}})^n)$, which is the collection of sign-invariant sets of F , is a regular stratification.

4.2. Infinitesimals

We will make extensive use of extensions of real fields by infinitesimals. This process is simple to implement computationally, and has been well formalized in [1] using the real spectrum. An elementary description of the use of infinitesimal elements is given in [4] in an algorithm for the existential theory of the reals.

One disadvantage of working over an infinitesimal extension field is that basic field operations become very expensive. Typically, an element of $\mathbb{R}(\epsilon, \delta)$ is represented as a polynomial in ϵ and δ . The degree of such elements will typically be $O(d^{O(k)})$, and clearly with 3 or 4 infinitesimals, each field operation is enormously expensive.

But in [6] a method is described for computing with infinitesimals which costs only slightly more than integer arithmetic in typical cases. The idea is to do arithmetic using straight line programs, and recover only the lowest degree rational coefficient of the field element by differentiation. Thus the use of infinitesimals in quantifier elimination can be a practical proposition.

DEFINITION 4.4. For a given real field R , we say that an element ϵ is infinitesimal with respect to R if the extension $R(\epsilon)$ is ordered such that ϵ is positive, but smaller than any positive element of R .

We will have cause to make use of towers of such field extensions. We will use the suggestive notation $\delta \gg \epsilon$ for two infinitesimals to mean that ϵ is infinitesimal with respect to the real closure of the field $R(\delta)$.

4.3. Transformation Algorithm

The following algorithm takes the formula B defining an arbitrary semi-algebraic set S and transforms it to a new formula B_0 defining a compact set, regularly stratified by the signs of polynomials defining it. The set S_0 defined by B_0 is non-empty if and only if S is. The size of B_0 is larger than B by at most a constant factor (2 if B is in standard form, 4 otherwise), and only a constant number of infinitesimals are needed (in the randomized version).

- Convert the input formula to monotone standard form.
- Add to the formula a conjunction with the polynomial inequality $\sum_{j=1}^k x_j^2 \leq 1/\rho^2$ (converted to standard form), where ρ is an infinitesimal. The resulting formula defines a bounded set in the extension field $\mathbb{R}(\rho)$.
- Choose an $a \in (\mathbb{R}_+)^n$ at random, or let $a_1 \gg a_2 \gg \dots \gg a_n > 0$ be a series of infinitesimals.
- Construct a formula B_0 from the input formula $B(A_1, \dots, A_n)$ as follows. For each atomic predicate A_i , replace A_i with

$$\begin{aligned} \text{if } A_i \text{ is } f_i = 0 \text{ then } & (f_i + \epsilon a_i \geq 0) \wedge (f_i - \epsilon a_i \leq 0) \\ \text{if } A_i \text{ is } f_i > 0 \text{ then } & (f_i - \delta a_i \geq 0) \end{aligned} \tag{21}$$

where $\delta \gg \epsilon$ are infinitesimals. Then the set S_0 defined by this formula is closed and bounded, therefore compact. By the results of [6] the connected components of the sign-invariant sets of S_0 (but not of S_0 itself, that requires two more infinitesimals) are in one-to-one correspondence with those of S . Return B_0 .

The correctness of the steps of the transformation was proved in [6].

4.4. Non-Emptiness for Compact, Regular Sets

Once we know that the set S_0 is compact and regularly stratified by the signs of polynomials f_i in B_0 , it is easy to decide if it is non-empty. We choose a linear map $\pi : \mathbb{R}^k \rightarrow \mathbb{R}$ and find all the critical points of this map restricted to the strata of S_0 . If S_0 is non-empty, π will attain a maximum value on it at some point P , which will be a critical point. By determining the signs of the polynomials f_i at some candidate critical point P , and

evaluating the predicate B_0 , we can check if P in fact lies in S_0 .

By testing all critical points in this way, we are guaranteed to find a “witness” P to $S_0 \neq \emptyset$ if it exists. Conversely, if S_0 is empty, no critical point will satisfy the formula.

4.5. Enumerating Critical Points

Since we know the polynomials f_i are in general position, the intersection of any $k + 1$ of them in k dimensions will be null. Any $j \leq k$ of them will intersect in a manifold of dimension $k - j$. Let P be an extremal point of π in the set S_0 . Then P is also an extremal of π in a manifold M which is the set of zeros of some polynomials f_{i_1}, \dots, f_{i_j} . These polynomials are precisely the f_i which are zero at P .

So to enumerate all potential witness points, we enumerate the critical points of π on the set of common zeros of f_{i_1}, \dots, f_{i_j} , for every set of $j \leq k$ polynomials. First we define a polynomial

$$g = \sum_{l=1}^j f_{i_l}^2$$

and solve the system

$$g = \mu \quad \frac{\partial g}{\partial x'_2} = \dots = \frac{\partial g}{\partial x'_k} = 0 \tag{22}$$

“in the limit” as $\mu \rightarrow 0$. The coordinates x'_2, \dots, x'_k are a basis for the $k - 1$ dimensional linear space which is the kernel of π . The process of solving this system in the limit is described in [4] and [6], and involves computing the u-resultant of the system, arranging it in powers of μ , and retaining the lowest degree coefficient. The result is a polynomial $p(s)$ and rational functions $(r_1(s), \dots, r_k(s))$ such that the solutions to the system (22) are all the tuples $(r_1(\alpha_i), \dots, r_k(\alpha_i))$ where α_i are the roots of $p(s) = 0$.

To compute the signs of the other polynomials at these critical points, we substitute $x_i \mapsto r_i(s)$ for $i = 1, \dots, k$, giving $q_i(s) = f_i(r(s))$ and the set of signs we are looking for is precisely the sign sequences of

$$(q_1(s), \dots, q_n(s)) \text{ at roots of } p(s) = 0$$

and to find these we simply apply the sign determination algorithm of the last section (to numerator and denominator, since the q_i here are rational functions).

4.6. Existential Decision Algorithm

Given an arbitrary formula $B(A_1, \dots, A_n)$, we first apply the transformation algorithm described earlier to produce a new formula B_0 defining a compact, regularly stratified set S_0 . Then we proceed as follows:

- Choose a generic linear map $\pi : \mathbb{R}^k \rightarrow \mathbb{R}$ by either selecting k random integers π_1, \dots, π_k , or for the deterministic version, by choosing infinitesimal $\pi_1 \gg \dots \gg \pi_k > 0$.

- Enumerate all subsets of $j \leq k$ polynomials $\{f_{i_1}, \dots, f_{i_j}\}$. Do this in order of increasing j , so that “easy” witness points will be found early.
- For each subset, construct a representation of the critical points of π as a polynomial $p(s)$ and rational functions $r_1(s), \dots, r_k(s)$.
- Substitute $r_i(s)$ for x_i in the other polynomials, giving $q_i(s) = f_i(r(s))$ for $i = 1, \dots, n$. Determine the signs of the $q_i(s)$ at roots of $p(s) = 0$ using the algorithm of the last section.
- Substitute these signs into the formula B_0 to check if the corresponding critical point lies in S_0 . If yes, S_0 is non-empty, so return “true”. If not, continue until no more critical points, and then return “false”.

4.7. Complexity Analysis

First, observe that the transformation algorithm changes only n by a constant factor, and leaves k , d and c unchanged. Clearly it can be done in linear time in these parameters.

The total number of $j \leq k$ -tuples of polynomials is $O(n^k)$. Constructing the representation for the solutions of the system (22) can be done in time $d^{O(k)}$, [4]. Applying Bezout’s theorem to this system, we see that the polynomial $p(s)$ has degree $O((2d)^k)$, and the numerator and denominator of the r_i ’s have the same bound. Substituting r_i ’s into the other f_i ’s gives n polynomials of degree $O(d(2d)^k)$ in s .

The sign determination step, using the bounds of the last section, takes $O(nk^3d(2d)^{2k})$ arithmetic steps. Substituting each sign sequence and evaluating the predicate B_0 takes $O(n)$ time. Multiplying by the number of polynomial tuples, we get an overall bound of

$$n^{(k+1)}d^{O(k)}$$

arithmetic steps over the coefficient field. To get the bit complexity, we must take into account the complexity of calculations over this field. This depends on whether we are using the randomized or deterministic version of the algorithm.

4.7.1. Randomized Version

For the randomized version, we must take into account the bit length of the randomly chosen a_i ’s and coordinates π_1, \dots, π_k of the projection map. We could try to figure out explicitly the conditions for a particular a, π combination to be a good choice, but there is a simpler argument we can use, which takes advantage of the fact that our calculation can be expressed as an algebraic decision tree. A particular (a, π) must be a good choice if *all the query polynomials in the decision tree are non-zero* at that value (excepting query polynomials which are identically zero, which can be ignored). This follows because for such an (a, π) , there is an open, connected neighborhood $N(a, \pi)$ such that all the query polynomials have the same sign over all of $N(a, \pi)$ as

they do at (a, π) . Thus the algorithm’s output is that same for all these choices. But almost all of the points in $N(a, \pi)$ must be good choices, since good points are dense. The algorithm must produce the correct output at these points, hence it produces the correct output at (a, π) .

So it suffices to choose (a, π) to avoid the zero sets of all the query polynomials. The query polynomials have degree $d^{O(k)}$ in field elements (and they are integral over the input) and there are potentially $n^{(k+1)}d^{O(k)}$ of them. The union of the zero sets gives us a bad set in the space $\mathbb{R}^{(n+k)}$ of possible (a, π) -values which has degree $(nd)^{O(k)}$. By Schwartz’s lemma, we will have probability p of hitting the bad set if we choose the a_i ’s and π_j ’s randomly with $\log(p^{-1}(nd)^{O(k)})$ bits. Fixing p , we see that $O(k \log(nd))$ bits suffice. For any $\epsilon > 0$, the, the overall bit complexity is therefore bounded by

$$n^{(k+1)}d^{O(k)}(c \log n)^{(1+\epsilon)} \quad (23)$$

where c is the length of the input coefficients in bits. This assumes that $O(b^{(1+\epsilon)})$ is the bound for arithmetic on b -bit integers.

This bound still conceals many constant factors. We must pay attention to the cost of the arithmetic over the field extension $\mathbb{Q}[\epsilon, \delta, \mu, \rho]$. This is of course, a polynomial which is already taken account in the bound above, but the actual time may still be prohibitive. As explained in [6] however, in most situations, we can perform this arithmetic for about the same cost as integer arithmetic.

4.7.2. Deterministic Version

For the deterministic version, we assume a and π are defined using infinitesimals. Specifically, we assume that

$$\rho \gg a_1 \gg \dots \gg a_n \gg \delta \gg \epsilon \gg \pi_1 \gg \dots \gg \pi_k \gg \mu > 0$$

are all infinitesimals. The order is forced by the sequence of steps in bounding, desingularization, compactification, and computing projections.

Since the query polynomials in the Sturm algorithm have degree $d^{O(k)}$ as polynomials in the input coefficients, and there are $O(n+k)$ distinct infinitesimals in the coefficient field, the naive bound for the size of a query polynomial would be $d^{O(nk+k^2)}$, assuming field elements are represented as polynomials in the infinitesimal elements with rational coefficients. Fortunately, though, we can get by with queries that depend on only $O(k)$ infinitesimals.

Observe first that in step three of the decision algorithm of section 4.6., $p(s)$ depends on $\leq k$ of the f_i ’s and the coordinates of π , and therefore on $O(k)$ infinitesimals total. The same holds true for the r_i ’s. If we can distinguish the roots of $p(s)$ and then we can enumerate the sign sequences by making Sturm queries that involve only one other f_i at a time.

This can be done either with the Sturm sequence of $p(s)$, or with the signs of all its derivatives. Let

$p_0(s), \dots, p_N(s)$ denote one of those sequences. Then the signs of $p_0(s), \dots, p_N(s)$ are distinct at each distinct root of $p(s) = 0$.

So for $i = 1, \dots, n$, we make separate calls to the univariate sign determination algorithm to get the signs of each sequence $(p_0(s), \dots, p_N(s), f_i(r(s)))$ at roots of $p(s) = 0$. The signs of $(p_0(s), \dots, p_N(s))$ are distinct at distinct roots of $p(s) = 0$, and in fact uniquely determine the order of those roots. So we can sort the sign sequences of $(p_0(s), \dots, p_N(s), f_i(r(s)))$ in order of these roots. After doing this for each f_i , and identifying those sign sequences with the same signs of $(p_0(s), \dots, p_N(s))$, we obtain all the signs of $(p_0(s), \dots, p_N(s), f_1(r(s)), \dots, f_n(r(s)))$ which is what we need. The number of arithmetic steps (over the coefficient field) in each call to the sign determination algorithm, since we have $d^{O(k)}$ polynomials of degree $d^{O(k)}$, using the bound in equation (19) is $d^{O(k)}$. Since we are making n calls to the sign determination algorithm, the complexity is $nd^{O(k)}$, as it was before.

Now since only $p(s)$, the sequence $(p_1(s), \dots, p_N(s))$ and one f_i appear in each call to the sign determination algorithm, the total number of infinitesimals in each call is $O(k)$. Since the Sturm queries have degree $d^{O(k)}$ over the base field, and we have $O(k)$ infinitesimals in the base field, the cost per arithmetic step is $d^{O(k^2)}$ integer operations. Multiplying this cost by the number of arithmetic steps in the algorithm overall, and assuming $b^{(1+\epsilon)}$ complexity for multiplying b -bit integers, the total bit complexity of the deterministic algorithm is

$$n^{(k+1)}d^{O(k^2)}c^{(1+\epsilon)} \tag{24}$$

for any $\epsilon > 0$.

4.8. Perspective

There are three reasons why this method offers better practical efficiency than other methods. The first is the way it deals with infinitesimals, as described in [6]. The second is in the form of the bound, which has a ‘‘combinatorial’’ part n^{k+1} and an algebraic part $d^{O(k)}$. The algebraic part is the same as the methods of [14], [21] and [12] although the constant hidden in the exponent is slightly better in our method because of the use of improved sign determination algorithm of the last section.

A more important difference is in the combinatorial part. The other schemes achieve combinatorial complexities only of $n^{O(k)}$, where the hidden constant is the same as for the algebraic part, and is at least 4. In typical geometric problems, n is large compared to d and k , so n^{k+1} is much smaller than n^{4k} . e.g. If $n = 10$, $k = 3$, we have $n^{k+1} = 10^4$, but $n^{4k} = 10^{12}$. The lower bound for the combinatorial complexity is n^k , so we have a near-optimal upper bound.

Note that the algebraic complexity can be improved to $O(n^k \log nd^{O(k)})$ using the techniques from [3]. Since all the witness points lie on ‘‘silhouette curves’’, one simply enumerates the curves, sorts the points along

them as in [3], and then re-evaluates the predicate while walking along the curve. No recursive calling is needed in this stripped down version of [3], so the algebraic complexity drops from $d^{O(k^2)}$ to $d^{O(k)}$. On the other hand, this would be a *less* practical algorithm than the one described here, because the constant factors in the $O(k)$ are larger, which will normally swamp the factor of n difference.

Finally, although we do not discuss it here, recent work on sparse resultants [7], [8] make it possible to eliminate large numbers of variables in a reasonable amount of time. The issue of the elimination method is often neglected in real quantifier elimination work, because bounds of the form $d^{O(k)}$ are in some sense optimal. However, the constant factor in the exponent can cause many orders of magnitude difference in running time, and in practice many polynomial systems have far fewer solutions than is predicted by the Bezout bound. The sparse resultant methods exploit this, as do Gröbner basis methods. However, Gröbner methods suffer from either uncontrolled coefficient growth over the real numbers (conventional Gröbner algorithms), or must sacrifice sparseness (enumerating syzygies by degree). In either case, they are much slower than sparse resultant algorithms when working with real coefficients [16], [17]. The sparse resultant is a key to the practical viability of our quantifier elimination algorithm.

5. CONCLUSIONS

In this paper we described a new algorithm for determining the signs of a collection of polynomials $q_1(s), \dots, q_n(s)$ at the roots of a polynomial $p(s) = 0$. The idea of the algorithm was to find a certain minimal submatrix of the Hadamard matrix H_n which was sufficient for the sign sequence calculations. We showed that such a matrix exists, denoted $K_{n,m}$, and that it has a pseudo-polynomial number of rows. We also showed that each Sturm query involves at most a logarithmic number of q_i 's. Finally we described our method for sign determination, the heart of which is an algorithm for computing a non-singular submatrix of $K_{n,m}$ directly from a list of column indices.

Our algorithm has better sequential complexity than the original BKR method, and it has the important property that the number of potential Sturm queries is pseudo-polynomial, rather than exponential as in BKR. This latter property allows us to do quantifier elimination for formulae constructed from polynomials with arbitrary coefficients.

Finally, we obtained a simple, practical, quantifier elimination algorithm by combining the sign-determination algorithm and a perturbation scheme with a small number of infinitesimals in its randomized version. This algorithm is being implemented with sparse resultants and straight-line programs to minimize the cost of computing with infinitesimals.

REFERENCES

- [1] J. Bochnak, M. Coste, and M-F. Roy. *Géométrie algébrique réelle*. Number 12 in *Ergebnisse der Mathematik 3*. Springer-Verlag, Berlin, 1987.
- [2] M. Ben-Or, D. Kozen, and J. Reif. The complexity of elementary algebra and geometry. *J. Comp. and Sys. Sci.*, 32:251–264, 1986.
- [3] J.F. Canny. *The Complexity of Robot Motion Planning*. M.I.T. Press, Cambridge, 1988.
- [4] J.F. Canny. Some algebraic and geometric computations in PSPACE. In *ACM Symposium on Theory of Computing*, pages 460–467, 1988.
- [5] J.F. Canny, D.Y. Grigor’ev and N.N. Vorobjov. Finding Connected Components of a Semialgebraic Set in Subexponential Time. *Applicable Algebra in Engineering, Communication and Computing*, 2:217–239, 1992.
- [6] J. Canny. Computing Roadmaps of General Semi-Algebraic Sets. *Computer Journal*, (this issue), 1993.
- [7] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Conference on Algebraic Algorithms and Error-Correcting Codes*, Puerto Rico, 1993.
- [8] I. Emiris and J. Canny. A practical algorithm for the sparse mixed resultant. In *International Symposium on Symbolic and Algebraic Computation*, 1993.
- [9] G. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Springer LNCS*, 33:135–183, 1975.
- [10] D.Y. Grigor’ev. Complexity of deciding Tarski algebra. *Journal of Symbolic Computation*, 5:65–108, 1988.
- [11] D.Y. Grigor’ev and N.N. Vorobjov. Solving systems of polynomial equations in subexponential time. *Journal of Symbolic Computation*, 5:37–64, 1988.
- [12] D.Y. Grigor’ev and N.N. Vorobjov. Counting connected components of a semialgebraic set in subexponential time. *Computational Complexity*, 2:133–186, 1992.
- [13] C.G. Gibson, K. Wirthmüller, A.A. Du Plessis, and E.J.N. Looijenga. *Topological Stability of Smooth Mappings*. Number 552 in *Lecture Notes in Mathematics*. Springer-Verlag, New York, 1976.
- [14] J. Heintz, M.F. Roy, and P. Solerno. Complexité du principe de Tarski-Seidenberg. *Comptes Rendu de l’Acad. de Sciences Paris*, 309:825–830, 1989.
- [15] J. Heintz, M.F. Roy, and P. Solerno. Single-exponential path finding in semialgebraic sets. In *Conference on Algebraic Algorithms and Error-Correcting Codes*, 1990.
- [16] D. Manocha and J.F. Canny. The implicit representation of rational parametric surfaces. *Journal of Symbolic Computation*, 13:485–510, 1992.
- [17] D. Manocha and J.F. Canny. Real time inverse kinematics of general 6R manipulators. In *IEEE Conference on Robotics and Automation*, pages 383–389, 1992.
- [18] C.A. Neff. Specified precision polynomial root isolation is in NC. In *IEEE Conference on Foundations of Computer Science*, 1990.
- [19] Paul Pedersen. Multivariate Sturm theory. In *Proc. AAECC-9, New Orleans*, Springer Lect. Notes in Comp. Sci. **539**. 1991.
- [20] P. Pedersen, M.-F. Roy, and A. Szpirglas. Counting real zeros in the multivariate case. In *Proc. MEGA-92*, 1992. to appear in ‘Computational Algebraic Geometry’, edited by F. Eyssette, and A. Galligo, Birkhauser.
- [21] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, parts I, II and III. In *Journal of Symbolic Computation*, 13(3):255–352, 1992.
- [22] M.F. Roy and A. Szpirglas. Complexity of computation on real algebraic numbers. *Journal of Symbolic Computation*, 10(1):39–52, 1990.
- [23] A. Seidenberg. Constructions in algebra. *Trans. AMS*, 197:273–313, 1974.
- [24] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 1948.