Subtropical Real Root Finding

Thomas Sturm

Max Planck Institute for Informatics

Saarbrücken, Germany

`sturm@mpi-inf.mpg.de`

January 20, 2015

Abstract

We describe a new incomplete but terminating method for real root finding for large multivariate polynomials. We take an abstract view of the polynomial as the set of exponent vectors associated with sign information on the coefficients. Then we employ linear programming to heuristically find roots. There is a specialized variant for roots with exclusively positive coordinates, which is of considerable interest for applications in chemistry and systems biology. An implementation of our method combining the computer algebra system Reduce with the linear programming solver Gurobi has been successfully applied to input data originating from established mathematical models used in these areas. We have solved several hundred problems with up to more than 800 000 monomials in up to 10 variables with degrees up to 12. Our method has failed due to its incompleteness in less than 8 percent of the cases.

1 Introduction

Our work discussed here is motivated by our studies of Hopf bifurcations [15, 14] for reaction systems in chemistry and gene regulatory networks in systems biology, which are originally given by systems of ordinary differential equations. Hopf bifurcations can be described algebraically [7, 32, 11, 10], resulting in one very large multivariate polynomial equation $f = 0$ subject to few much simpler polynomial side conditions $g_1 > 0, \ldots, g_n > 0$. For such systems one is interested in feasibility over the reals and, in the positive case, in at least one feasible point. It turns out that, generally, scientifically meaningful information can be obtained already by checking only the feasibility of $f = 0$, which is the focus of this article. For further details on the scientific background, we refer the reader to our publications [28, 29, 33, 9, 8].

With one of our models, viz. *Mitogen-activated protein kinase (MAPK)*, we obtain and solve polynomials of considerable size. Our currently largest instance `mapke5e6` contains 863 438 monomials in 10 variables. One of the variables occurs with degree 12, all other variables occur with degree 5. Such problem sizes are clearly beyond the scope of classical methods in symbolic computation. To give an impression, the size of an input file with `mapke5e6` in infix notation is 30 MB large. LaTeX-formatted printing of `mapke5e6` would fill more than 3000 pages in this document. The MAPK model actually yields even larger instances,

which we, unfortunately, cannot generate at present, because in our toolchain Maple cannot produce polynomials larger than 32 MB.

This article introduces an incomplete but terminating algorithm for finding real roots of large multivariate polynomials. The principle idea is to take an abstract view of the polynomial as the set of its exponent vectors supplemented with sign information on the corresponding coefficients. To that extent, out approach is quite similar to tropical algebraic geometry [30]. However, after our abstraction we do not consider tropical varieties but employ linear programming to determine certain suitable points in the Newton polytope, which somewhat resembles successful approaches to sum-of-square decompositions [26].

We have implemented our algorithm in Reduce [16] using direct function calls to the dynamic library of the LP solver Gurobi [13]. In practical computations on several hundred examples, our method has failed do to its incompleteness in less than 8 percent of the cases. The longest computation time observed was around 16 s. As mentioned above, the limiting factor at present is the technical generation of even larger input.

In Section 2 we introduce a specialization of our method that only finds roots with all positive coordinates. This is highly relevant in our context of reaction networks, where typically all variables are known to be positive. We also discuss an illustrating example in detail. Section 3 generalizes our method to arbitrary roots. In Section 4 we discuss issues and share experiences related to a practical implementation of our method. In Section 5 we evaluate the performance of our method with respect to efficiency and to its incompleteness on several hundred examples originating from four different chemical and biological models.

2 Finding Roots with Positive Coordinates

Denote $\mathbb{N}_1 = \mathbb{N} \setminus \{0\}$, and let $d \in \mathbb{N}_1$. For $a \in \mathbb{R}$, vectors $\mathbf{x} = (x_1, \ldots, x_d)$ of either indeterminates or real numbers, and $\mathbf{p} = (p_1, \ldots, p_d) \in \mathbb{N}^d$, we use the notations $a^{\mathbf{p}} = (a^{p_1}, \ldots, a^{p_d})$ and $\mathbf{x}^{\mathbf{p}} = x_1^{d_1} \cdots x_d^{p_d}$. We will, however, never consider a vector to the power of a number. Our notations are compatible with the standard scalar product as follows:

$$(a^{\mathbf{p}})^{\mathbf{q}} = (a^{p_1}, \ldots, a^{p_d})^{\mathbf{q}} = a^{p_1 q_1} \cdots a^{p_d q_d} = a^{\mathbf{p}\mathbf{q}}.$$

Consider a multivariate integer polynomial

$$f = \sum_{\mathbf{p} \in \mathrm{supp}(f)} \mathrm{coeff}(f, \mathbf{p}) \cdot \mathbf{x}^{\mathbf{p}} \in \mathbb{Z}[\mathbf{x}],$$

where $\mathrm{coeff}(f, \mathbf{p}) \neq 0$ for $\mathbf{p} \in \mathrm{supp}(f)$, which is called the *support* of f.

2.1 Finding a Point with Positive Value

The *Newton polytope* of f is the convex hull of $\mathrm{supp}(f)$. It forms a polyhedron in $\mathbb{R}^d$, which we identify with its vertices, formally $\mathrm{newton}(f) \subseteq \mathrm{supp}(f)$. The following lemma is a straightforward consequence of the convex hull property.

Lemma 1. Let $f = \mathrm{coeff}(f, \mathbf{p}) \cdot \mathbf{x}^{\mathbf{p}} + f' \in \mathbb{Z}[\mathbf{x}]$. Assume that $\mathbf{p} \notin \mathrm{newton}(f)$. Then $\mathrm{newton}(f) = \mathrm{newton}(f')$. $\square$

For $\mathbf{p} \in \text{supp}(f)$ we define $\text{sign}(f, \mathbf{p}) = \text{sign}(\text{coeff}(f, \mathbf{p}))$. We partition the support of f as follows:

$$
\begin{aligned}
\text{supp}(f) &= \text{supp}^+(f) \,\dot\cup\, \text{supp}^-(f) \,\dot\cup\, \text{supp}^0(f), \\
\text{supp}^+(f) &= \{\, \mathbf{p} \in \text{supp}(f) \mid \text{sign}(f, \mathbf{p}) > 0 \wedge \mathbf{p} \neq \mathbf{0} \,\}, \\
\text{supp}^-(f) &= \{\, \mathbf{p} \in \text{supp}(f) \mid \text{sign}(f, \mathbf{p}) < 0 \wedge \mathbf{p} \neq \mathbf{0} \,\}, \\
\text{supp}^0(f) &= \text{supp}(f) \cap \{\mathbf{0}\}.
\end{aligned}
$$

Let $\text{supp}^+(f) = \{\mathbf{p}_1, \ldots, \mathbf{p}_r\}$, $\text{supp}^-(f) = \{\mathbf{p}_{r+1}, \ldots, \mathbf{p}_s\}$, and fix any order on $\text{supp}(f)$. The *basic LP matrix* $B(f)$ is composed as follows, where the last row is present if and only if $\text{supp}^0(f) \neq \emptyset$:

$$
B(f) = \left[
\begin{array}{c}
B^+(f) \\
\hline
B^-(f) \\
\hline
(\mathbf{0}, -1)
\end{array}
\right]
=
\left[
\begin{array}{cccc}
p_{11} & \cdots & p_{1d} & -1 \\
\vdots & \ddots & \vdots & \vdots \\
p_{rd} & \cdots & p_{rd} & -1 \\
\hline
p_{r+1,1} & \cdots & p_{r+1,d} & -1 \\
\vdots & \ddots & \vdots & \vdots \\
p_{s,d} & \cdots & p_{s,d} & -1 \\
\hline
0 & \cdots & 0 & -1
\end{array}
\right].
$$

Considering matrices concatenations of their rows, we write this also as $B(f) = B^+(f) \circ B^-(f) \circ (\mathbf{0}, -1)^*$. Whenever we write for a given matrix $B \in \mathbb{Z}^{m \times n}$ a product $N^* B$, then we implicitly agree that

$$
N^* = \left[
\begin{array}{ccccc}
-1 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & \cdots \\
0 & 0 & 1 & 0 & \cdots \\
\vdots & \vdots & \ddots & \ddots & \ddots
\end{array}
\right] \in \mathbb{Z}^{m \times m}.
$$

That is, the multiplication $N^* B$ replaces the elements of the first row of B with their additive inverses. Similarly, $-\mathbf{1} = (-1, \ldots, -1)^T$ is generally a column matrix of suitable length. In these terms, we are going to consider systems

$$
N^* \cdot B(f) \cdot \mathbf{x}^T \leq -\mathbf{1}, \quad \text{where} \quad \mathbf{x} = (\mathbf{n}, c) \in \mathbb{R}^{d+1},
$$

which can be rewritten as follows:

$$
\begin{aligned}
\mathbf{p}_1 \mathbf{n} - c &\geq 1 \\
\mathbf{p}_i \mathbf{n} - c &\leq -1, \quad i \in \{2, \ldots, s\}.
\end{aligned}
$$

Lemma 2. Let $f \in \mathbb{Z}[\mathbf{x}]$. Let $\mathbf{n} \in \mathbb{R}^d$, and let $c \in \mathbb{R}$. Then the following are equivalent:

(i) The hyperplane $H(\mathbf{x})$ defined by $\mathbf{n}\mathbf{x} = c$ strictly separates the point $\mathbf{p}_1$ from $\text{supp}(f) \setminus \{\mathbf{p}_1\}$, and the normal vector $\mathbf{n}$ is pointing from $H(\mathbf{x})$ in direction $\mathbf{p}_1$. In particular, $\mathbf{p}_1 \in \text{newton}(f)$.

(ii) There is $0 < \lambda \in \mathbb{R}$ s.t. $N^* \cdot B(f) \cdot (\lambda \mathbf{n}, \lambda c)^T \leq -\mathbf{1}$.

Proof. Assume that (i) holds. The orientation of $\mathbf{n}$ is chosen such that $\mathbf{n} \cdot \mathbf{p}_1 > c$ and $\mathbf{n} \cdot \mathbf{p}_i < c$ for $i \in \{2, \ldots, s\}$. Define $\delta = \min_{i \in \{1, \ldots, s\}} |\text{dist}(\mathbf{p}_i, H)| > 0$. Then

$$
\begin{aligned}
\mathbf{p}_1 \cdot \mathbf{n} - c &\geq \delta \|\mathbf{n}\|, \\
\mathbf{p}_i \cdot \mathbf{n} - c &\leq -\delta \|\mathbf{n}\|, \qquad i \in \{2, \ldots, s\},
\end{aligned}
$$

and we can choose $\lambda = (\delta \|\mathbf{n}\|)^{-1}$.

Vice versa, assume that (ii) holds. It follows that

$$
\begin{aligned}
\mathbf{p}_1 \cdot \mathbf{n} &\geq c + 1/\lambda, \\
\mathbf{p}_i \cdot \mathbf{n} &\leq c - 1/\lambda, \qquad i \in \{2, \ldots, s\}.
\end{aligned}
$$

Hence $H(\mathbf{x})$ defined by $\mathbf{n}\mathbf{x} = c$ is a hyperplane separating $\mathbf{p}_1$ from $\text{supp}(f) \setminus \{\mathbf{p}_1\}$, where the distance between $H(\mathbf{x})$ and $\text{supp}(f)$ is at least $\|\mathbf{n}\|/\lambda > 0$. Furthermore, $\mathbf{n}$ is oriented as required in (i). $\square$

Lemma 3. Let $0 \neq f \in \mathbb{Z}[\mathbf{x}]$. Then the following are equivalent:

(i) There is $(\mathbf{n}, c) \in \mathbb{R}^{d+1}$ s.t. $N^* \cdot B(f) \cdot (\mathbf{n}, c)^T \leq -\mathbf{1}$.

(ii) There is $(\mathbf{n}, c) \in \mathbb{Q}^{d+1}$ s.t. $N^* \cdot B(f) \cdot (\mathbf{n}, c)^T \leq -\mathbf{1}$.

(iii) There is $\mathbf{n} \in \mathbb{Z}^d$, $c \in \mathbb{Q}$ s.t. $N^* \cdot B(f) \cdot (\mathbf{n}, c)^T \leq -\mathbf{1}$.

Proof. The existence of a real solution in (i) and a rational solution in (ii) coincide due to the Linear Tarski Principle: Ordered fields admit quantifier elimination for linear formulas [21]. Given a solution $(n_1, \ldots, n_d, c) \in \mathbb{Q}^{d+1}$ in (ii), we can use the principal denominator $m \in \mathbb{N}_1$ of $n_1, \ldots, n_d$ to obtain a solution $(mn_1, \ldots, mn_d, mc + m - 1) \in \mathbb{Z}^d \times \mathbb{Q}$ in (iii). The implication from (iii) to (i) is trivial. $\square$

Lemma 4. Let $f \in \mathbb{Z}[\mathbf{x}] \setminus \mathbb{Z}$. Let $(\mathbf{n}, c) \in \mathbb{R}^{d+1}$ such that $N^* \cdot B(f) \cdot (\mathbf{n}, c)^T \leq -\mathbf{1}$. Then there is $a_0 \in \mathbb{N}$ such that for all $a \in \mathbb{N}$ with $a \geq a_0$ the following hold:

(i) $\left| \text{coeff}(f, \mathbf{p}_1) \cdot a^{\mathbf{n}\mathbf{p}_1} \right| > \left| \sum_{i=2}^{s} \text{coeff}(f, \mathbf{p}_i) \cdot a^{\mathbf{n}\mathbf{p}_i} \right|,$

(ii) $\text{sign}\big(f(a^{\mathbf{n}})\big) = \text{sign}(f, \mathbf{p}_1).$

Proof. (i) From $f \notin \mathbb{Z}$ it follows that $\mathbf{p}_1 \neq \mathbf{0}$. By Lemma 2 we know $\mathbf{n}\mathbf{p}_1 > c$ and $\mathbf{n}\mathbf{p}_i < c$ for $i \in \{2, \ldots, s\}$. It follows that there is $0 < \delta \in \mathbb{R}$ such that

$$
\begin{aligned}
\mathbf{n}\mathbf{p}_1 &\geq c + \delta, & (1) \\
\mathbf{n}\mathbf{p}_i &\leq c - \delta & i \in \{2, \ldots, s\}. & (2)
\end{aligned}
$$

We are going to show that $a_0 = \left\lceil \max\left\{2, \big(b \cdot (k-1)\big)^{\frac{1}{\delta}}\right\} \right\rceil$ is a suitable choice, where

$$
b = |\text{coeff}(f, \mathbf{p}_1)|^{-1} \cdot \max_{i \in \{2, \ldots, s\}} |\text{coeff}(f, \mathbf{p}_i)|.
$$

For $a \geq a_0 \geq 2$ and for all $i \in \{2, \ldots, s\}$, the inequalities (1) and (2) and monotony yield

$$
a^{\mathbf{n}\mathbf{p}_1} \geq a^\delta a^c > a^\delta a^c a^{-\delta} \geq a^\delta a^{\mathbf{n}\mathbf{p}_i} \geq b \cdot (k-1) \cdot a^{\mathbf{n}\mathbf{p}_i}.
$$

function find-positive(f)

> **data** : $f \in \mathbb{Z}[x_1, \ldots, x_d]$
> **result**: $\mathbf{p} \in (\mathbb{Q}^+)^d$ or `"failed"`

1    $B^+ := B^+(f)$
2    $B^- := B^-(f)$
3    $h := $ `"infeasible"`
4   **while** $h = $ `"infeasible"` *and* $B^+ \neq [\,]$ **do**
5      $h := $ `lpsolve`$(B^+ \circ B^- \circ (\mathbf{0}, -1)^*)$
6   delete the first row from B^+
7   **if** $h = $ `"infeasible"` **then**
8   **return** `"failed"`
9    $(\mathbf{n}, c) := h$
10    $t := 2$
11   **while** $f(t^{\mathbf{n}}) \leq 0$ **do**
12      $t := 2t$
13   **return** $t^{\mathbf{n}}$

function lpsolve(B)

> **data** : a matrix B with $d + 1$ columns
> **result**: $(\mathbf{n}, c) \in \mathbb{Z}^d \times \mathbb{Q}$ or `"infeasible"`

14    $\Pi := $ LP problem given by N^*B and $-\mathbf{1}$
15    $h := $ a solution $(\mathbf{n}, c) \in \mathbb{Q}^{d+1}$ of Π or `"infeasible"`
16   **if** $h = $ `"infeasible"` **then**
17   **return** `"infeasible"`
18    $m := $ principal denominator of the coordinates of $\mathbf{n}$
19    $\mathbf{n} := m \cdot \mathbf{n}$
20    $c := mc + m - 1$
21   **return** $(\mathbf{n}, c)$

Algorithm 1: Functions `find-positive` and `lpsolve`

Using the triangle inequality it follows that

$$a^{\mathbf{np}_1} > b \sum_{i=2}^{s} a^{\mathbf{np}_i} \geq |\mathrm{coeff}(f, \mathbf{p}_1)|^{-1} \cdot \left| \sum_{i=2}^{s} \mathrm{coeff}(f, \mathbf{p}_i) \cdot a^{\mathbf{np}_i} \right|,$$

which straightforwardly implies

$$|\mathrm{coeff}(f, \mathbf{p}_1) \cdot a^{\mathbf{np}_1}| > \left| \sum_{i=2}^{s} \mathrm{coeff}(f, \mathbf{p}_i) \cdot a^{\mathbf{np}_i} \right|.$$

(ii) It follows from (i) that for $a \geq a_0$ the sign of the monomial $\mathrm{coeff}(f, \mathbf{p}_1) \cdot a^{\mathbf{np}_1}$ determines the sign of $f(a^{\mathbf{n}})$. Since $a > 0$, we obtain

$$\mathrm{sign}\big(f(a^{\mathbf{n}})\big) = \mathrm{sign}\big(\mathrm{coeff}(f, \mathbf{p}_1) \cdot a^{\mathbf{np}_1}\big) = \mathrm{sign}(f, \mathbf{p}_1). \qquad \square$$

After these preparations we can state our first subalgorithm as Algorithm 1.

Theorem 5 (Correctness of `find-positive`). *Consider $f \in \mathbb{Z}[\mathbf{x}]$. Then the following hold:*

(i) The function `find-positive` terminates.

(ii) The function `find-positive` returns either `"failed"` or $\mathbf{p} \in (\mathbb{Q}^+)^d$ with $f(\mathbf{p}) > 0$.

Proof. (i) The termination of `lpsolve` follows from the existence of terminating algorithms for linear programming in line 15, including the Simplex algorithm [5], the ellipsoid method [19], and the interior point method [17]. For the function `find-positive` itself, the number of iterations of the while-loop in line 3 is bounded by the number of rows of B^+, which is in turn bounded by the finite cardinality of $\mathrm{supp}(f)$. The termination of the while-loop in line 11 will be discussed with the correctness in (ii).

(ii) To start with, the subroutine `lpsolve` solves the LP problem Π defined in line 14 and, in the feasible case, $(\mathbf{n}, c)$ in line 15 is a feasible point in $\mathbb{Q}^{d+1}$. The return value $(\mathbf{n}, c) \in \mathbb{Z}^d \times \mathbb{Q}$ in line 21 is a feasible point for Π as well. Its construction in lines 18–20 follows the proof step from (ii) to (iii) in Lemma 3.

Next, the while-loop in line 3 has the following loop invariants. Consider

$$f_{(n)} = f - \sum_{i=1}^{n-1} \mathrm{coeff}(f, \mathbf{p}_i)\mathbf{x}^{\mathbf{p}_i}$$

before the n-th iteration:

(I_1) $\mathrm{newton}(f_{(n)}) = \mathrm{newton}(f)$,

(I_2) $B\big(f_{(n)}\big) = B_{(n)}^+ \circ B^- \circ (\mathbf{0}, -1)^*$.

Invariant (I_2) is easy to see. Consider (I_1). For $n = 1$ this is trivial. Before the $n + 1$-st iteration we know that $h = \texttt{"infeasible"}$, which means that the LP problem given by

$$N^* \cdot \big((\mathbf{p}_n) \circ B_{(n+1)}^+ \circ B^- \circ (\mathbf{0}, -1)^*\big) \quad \text{and} \quad -\mathbf{1}$$

was infeasible at the n-th iteration. According to Lemma 2 it follows that $\mathbf{p}_n \notin \mathrm{newton}(f_{(n)})$. Using Lemma 1 and the induction hypothesis we conclude

$$\mathrm{newton}(f_{(n+1)}) = \mathrm{newton}(f_{(n)} - \mathrm{coeff}(f, \mathbf{p}_n)\mathbf{x}^{\mathbf{p}_n}) = \mathrm{newton}(f_{(n)}) = \mathrm{newton}(f).$$

The function `find-positive` has two possible exit points at lines 8 and 13 corresponding to its two possible return values. Assume we are in line 13. We have to show that $t^{\mathbf{n}} \in (\mathbb{Q}^+)^d$ with $f(t^{\mathbf{n}}) > 0$. The while-loop in line 3 has terminated after n iterations, and the if-condition in line 7 is false. In line 9 we know by (I_1), (I_2), and Lemma 2 that the feasible regions for $N^* \cdot (B^+ \circ B^- \circ (\mathbf{0}, -1)^*) \cdot \mathbf{v} \le -\mathbf{1}$ and $N^* \cdot B(f) \cdot \mathbf{v} \le -\mathbf{1}$ are identical. This allows us to use $B^+ \circ B^- \circ (\mathbf{0}, -1)^*$ instead of $B(f)$ for applying Lemma 4 to the original f, and our $\mathbf{n} \in \mathbb{Z}^d$ has the property described there. In line 11, at the beginning of the k-th iteration of the while-loop we have $t = 2^k$. By Lemma 4 we know that we will eventually have $t \ge a_0$ and thus $\mathrm{sign}(f(t^{\mathbf{n}})) = \mathrm{sign}(f, \mathbf{p}_n) > 0$. $\qquad\square$

```
     function find-zero(f)
         data  : f ∈ ℤ[x₁, ..., x_d]
         result: z ∈ (ℚ⁺)^d or "failed"
1        y := f(1)
2        if y = 0 then
3          │ return 1
4        if y > 0 then
5          │ f := -f
6        p := find-positive(f)
7        if q = "failed" then
8          │ return "failed"
9        z := construct-zero(f, p, 1)
10       return z

     function construct-zero(f, p, q)
         data  : f ∈ ℤ[x₁, ..., x_d], p, q ∈ ℚ^d
         result: z ∈ ℚ^d or "failed"
11       b := p + y · (q - p), where y is a new variable
12       g := f(b)
13       isolate r ∈ ]0, 1[ with g(r) = 0
14       z := b(r)
15       return z
```

Algorithm 2: Functions `find-zero` and `construct-zero`

2.2 Finding a Zero

We have discussed how to heuristically find $\mathbf{p} \in (\mathbb{Q}^+)^d$ such that $f(\mathbf{p}) > 0$ for our given $f \in \mathbb{Z}[\mathbf{x}]$. On that basis Algorithm 2 computes $\mathbf{z} \in (\bar{\mathbb{Q}}^+)^d$ such that $f(\mathbf{z}) = 0$, where $\bar{\mathbb{Q}}$ denotes the algebraic closure of $\mathbb{Q}$.

Lemma 6 (Correctness of `construct-zero`)**.** Consider $f \in \mathbb{Z}[\mathbf{x}]$, and let $\mathbf{p}$, $\mathbf{q} \in \mathbb{Q}^d$ such that $f(\mathbf{p})f(\mathbf{q}) < 0$. Then the following hold:

(i) The function `construct-zero` terminates.

(ii) The function `construct-zero` returns either `"failed"` or $\mathbf{z} \in \bar{\mathbb{Q}}^d$ with $f(\mathbf{z}) = 0$. If $\mathbf{p}$, $\mathbf{q} \in (\mathbb{Q}^+)^d$, then $\mathbf{z} \in (\bar{\mathbb{Q}}^+)^d$.

Proof. (i) The termination of `construct-zero` follows from the existence of terminating algorithms for univariate real root isolation including Sturm sequences [27] and more efficient algorithms [4, 1] based on Vincent's Theorem [31].

(ii) Since f is continuous and $f(\mathbf{p})f(\mathbf{q}) < 0$, the intermediate value theorem guarantees the existence of $\mathbf{z} \in \overline{\mathbf{p}\mathbf{q}}$ with $f(\mathbf{z}) = 0$. Formally, $\mathbf{z} \in \bar{\mathbb{Q}}^d$ is a solution

for $\mathbf{x}$ of the following nonlinear system with indeterminates $x_1, \ldots, x_d, y$:

$$f = 0 \tag{3}$$
$$x_1 = p_1 + y \cdot (q_1 - p_1) \tag{4}$$
$$\vdots$$
$$x_d = p_d + y \cdot (q_d - p_d) \tag{5}$$
$$y > 0 \tag{6}$$
$$y < 1. \tag{7}$$

In line 11, $\mathbf{b}$ is assigned the vector of the right hand sides of the d equations (4)–(5). In line 12, these are plugged into the left hand side of equation (3) yielding a nonlinear univariate polynomial equation in y. Using any of the methods mentioned in (i), we obtain in line 13 a solution $r \in \bar{\mathbb{Q}}$ for y of that equation subject to the constraints (6)–(7). That solution r is a real algebraic number in some suitable representation [23]. In line 14 we substitute r back into the equations (4)–(5) to finally obtain $\mathbf{x} = \mathbf{z} \in \bar{\mathbb{Q}}$, also as a real algebraic number.

Since $\mathbf{z} \in \overline{\mathbf{pq}}$, it follows from $\mathbf{p}, \mathbf{q} \in (\mathbb{Q}^+)^d$ that also $\mathbf{z} \in (\mathbb{Q}^+)^d$. $\qquad\square$

On the basis of Lemma 6 the following theorem is straightforward.

Theorem 7 (Correctness of `find-zero`). *Let $f \in \mathbb{Z}[\mathbf{x}]$. Then the function* `find-zero` *terminates and returns either* `"failed"` *or* $\mathbf{z} \in (\bar{\mathbb{Q}}^+)^d$ *with* $f(\mathbf{z}) = 0$.
$\qquad\square$

When one is interested only in the *existence* of a zero of f, then one can, in the positive case, obviously skip `construct-zero` and exit from `find-zero` after line 8. Notice that, in addition, one can then also exit early from `find-positive` after line 8 in Algorithm 1.

2.3 An Illustrating Example

Consider $f = -2x_1^5 + x_1^2 x_2 - 3x_1^2 - x_2^3 + 2x_2^2 \in \mathbb{Z}[x_1, x_2]$. We apply `find-zero` to find a point on the variety of f. Figure 1 pictures the variety. We obtain $f(1,1) = -3 < 0$, and apply `find-positive` to f.

Figure 2 pictures the support of f and indicates the Newton polytope. We split into $\mathrm{supp}^+(f) = \{(2,1),(0,2)\}$, $\mathrm{supp}^-(f) = \{(2,0),(5,0),(0,3)\}$, and $\mathrm{supp}^0(f) = \emptyset$, and we construct

$$B^+ = \begin{bmatrix} 2 & 1 & -1 \\ 0 & 2 & -1 \end{bmatrix}, \quad B^- = \begin{bmatrix} 2 & 0 & -1 \\ 5 & 0 & -1 \\ 0 & 3 & -1 \end{bmatrix}.$$

Our first LP problem

$$\begin{bmatrix} -2 & -1 & 1 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \\ 5 & 0 & -1 \\ 0 & 3 & -1 \end{bmatrix} \cdot (\mathbf{n}, c)^T \leq \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$
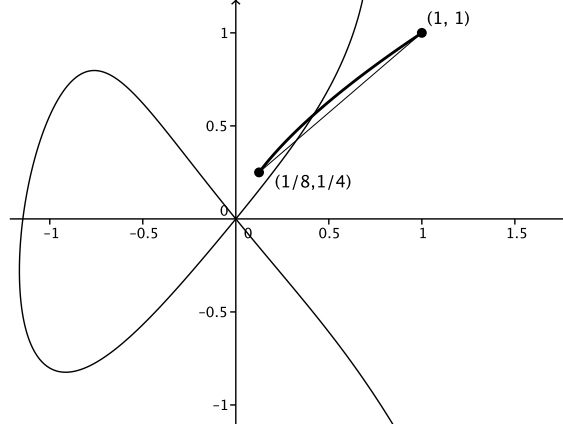
Figure 1: The variety of $f = -2x_1^5 + x_1^2 x_2 - 3x_1^2 - x_2^3 + 2x_2^2$ and the segment given by $t \in [0, 2]$ of the moment curve (t^{-3}, t^{-2}) corresponding to the normal vector $(-3, -2)$ of the separating hyperplane in Figure 2.
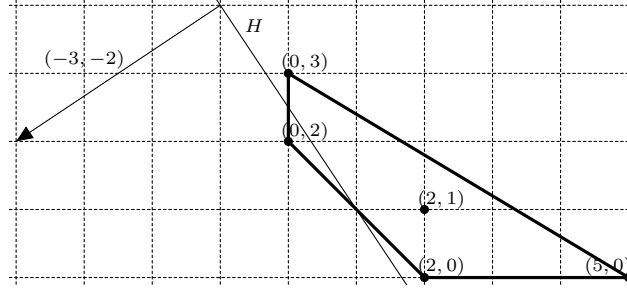

Figure 2: A subtropical view on $f = -2x_1^5 + x_1^2 x_2 - 3x_1^2 - x_2^3 + 2x_2^2$ from Figure 1. We see a hyperplane separating $(0, 2) \in \mathrm{newton}(f)$ from $\mathrm{supp}(f) \setminus \{(0, 2)\}$ together with its normal vector $\mathbf{n} = (-3, -2)$.

is infeasible, which confirms the observation in Figure 2 that $(2, 1) \notin \mathrm{newton}(f)$. Our next LP problem

$$\begin{bmatrix} 0 & -2 & 1 \\ 2 & 0 & -1 \\ 5 & 0 & -1 \\ 0 & 3 & -1 \end{bmatrix} \cdot (\mathbf{n}, c)^T \leq \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

is feasible with $\mathbf{n} = (-3, -2)$ and $c = 5$. Figure 2 shows the corresponding hyperplane H given by $-3x_1 - 2x_2 + 5 = 0$. It strictly separates $(0, 2) \in \mathrm{newton}(f)$ from $\mathrm{supp}(f) \setminus \{(0, 2)\}$, and its normal vector $\mathbf{n} = (-3, -2)$ is oriented towards $(0, 2)$. We now know that $f(t^{-3}, t^{-2}) > 0$ for sufficiently large positive t. In fact, already

$$f(2^{-3}, 2^{-2}) = f\left(\frac{1}{8}, \frac{1}{4}\right) = \frac{1087}{16384}.$$

9

The relevant part of the moment curve (t^{-3}, t^{-2}) for $t \in [1, 2]$ is pictured in Figure 1. Since both coordinates of **n** happen to be negative, the curve will for $t \to \infty$ not extend to infinity but converge to the origin. In particular, the curve will not leave the sign invariant region containing $\left(\frac{1}{8}, \frac{1}{4}\right)$.

Finally, we call `construct-zero` with $\left(\frac{1}{8}, \frac{1}{4}\right)$ and $(1, 1)$ to solve the system

$$
\begin{aligned}
-2x_1^5 + x_1^2 x_2 - 3x_1^2 - x_2^3 + 2x_2^2 &= 0 \\
x_1 &= \tfrac{1}{8} + y \cdot \left(1 - \tfrac{1}{8}\right) \\
x_2 &= \tfrac{1}{4} + y \cdot \left(1 - \tfrac{1}{4}\right) \\
y &> 0 \\
y &< 1.
\end{aligned}
$$

Dropping a positive integer denominator, we obtain the univariate polynomial

$$\bar{g} = -16807y^5 - 12005y^4 - 934y^3 - 20778y^2 + 285y + 1087$$

and an isolating interval $y \in]0.2, 0.3[$. Substitution of the real algebraic number $\langle \bar{g},]0.2, 0.3[\rangle$ into the equations for x_1 and x_2 yields an exact solution

$$
\begin{aligned}
x_1 &= \langle 686x^5 - 78x^3 + 584x^2 - 150x - 13,]0.32, 0.33[\rangle, \\
x_2 &= \langle 16807x^5 - 12005x^4 + 2026x^3 + 9122x^2 - 4609x + 323,]0.42, 0.43[\rangle,
\end{aligned}
$$

where the intervals can, of course, be refined to arbitrary precision. Geometrically, our solving has intersected the variety with the line segment connecting the end points of our moment curve segment, which is also indicated in Figure 1.

2.4 Why Strictly Positive Coordinates?

In the present section, we have focused on roots with strictly positive coordinates. This not only slightly simplifies the presentation. In fact, it is an important feature of our algorithm to be able to perform such a directed search.

To start with, the research presented here was originally motivated by questions on the stability of chemical and biological reaction networks, where the variables of the models typically are strictly positive. Our practical computations in Section 5 are taken from those areas. For details on the theoretical background we refer the reader to [3, 28, 29, 33, 9, 8].

Furthermore, the concept of positive feasible points is well-known from linear programming. Techniques used there can be straightforwardly transfered to our situation: Consider $f \in \mathbb{Z}[x_1, \ldots, x_d]$. For finding zeros $(z_1, \ldots, z_d)$ with $\text{sign}(z_i) = s_i \in \{-1, 1\}$ consider $f(s_1 x_1, \ldots, s_d x_d)$, for $z_1 \in]\alpha, \infty[$ consider $f(x_1 + \alpha, x_2, \ldots, x_d)$, for $z_1 \in]-\infty, \beta[$ consider $f(-x_1 - \beta, x_2, \ldots, x_d)$, and for x_1 unbounded consider $f(x_1 - x_1', x_2, \ldots, x_d)$ introducing a new variable x_1'.

3 Finding Arbitrary Roots

3.1 Using a Transformation

Consider $f \in \mathbb{Z}[x_1, \ldots, x_d]$. At the end of the previous section we have addressed a technique for turning a real feasibility test based on positive variables into a

general one. Using the observation that every real number is a difference of two positive real numbers, on introduces additional variables x'_1, ..., x'_d and transforms f into $f(x_1 - x'_1, \ldots, x_d - x'_d)$. That transformation is ubiquitous in linear programming, if not explicitly then implicitly within the solvers.

From an efficiency point of view our procedure is clearly dominated by the LP solving steps, where we have $d+1$ variables and $|\operatorname{supp}(f)|$ many constraints. Thinking in terms of state-of-the-art LP solvers [13, 22] and the Simplex method with the option of dualization [2, 20], the crucial complexity parameter is $\min\{d, |\operatorname{supp}(f)|\}$. With our considered transformation the cardinality of the support increases exponentially in d in the worst case, but the number of variables only doubles from d to $2d$.

Recall that our incomplete method relies on finding some $\mathbf{p} \in \operatorname{newton}(f)$ with $\operatorname{coeff}(f, \mathbf{p}) > 0$. We would like to point out that, doubling the dimension d with the transformation, the ratio $|\operatorname{newton}(f)| / |\operatorname{supp}(f)|$ will in general increase for geometric reasons [6]. Furthermore the exponential increase of $|\operatorname{supp}(f)|$ increases the absolute number of candidates for a suitable $\mathbf{p}$. On the other hand, the transformation does not add points to $\operatorname{supp}(f)$ but exchanges it entirely. It would require either comprehensive empirical studies or a thorough average-case analysis to make a precise statement about the quality of the transformation in terms of incompleteness.

3.2 A Genuine Generalization

We are now going to describe a generalization of the function `find-positive` in Algorithm 1, which searches for a suitable $\mathbf{p}$ not only in $\operatorname{supp}^+(f)$ but also in a subset of $\operatorname{supp}^-(f)$. Recall that in case of success, `find-positive` identifies the first row of B^+ as corresponding to the exponent vector of a monomial with positive coefficient that dominates f in the sense of Lemma 4. Then it constructs a point $\mathbf{p}$ with large suitably balanced positive coordinates. The key idea for our generalization is the following: If the coefficient of an otherwise suitable monomial is negative but there is at least one odd exponent in the exponent vector, then we can correct the "wrong" sign of the coefficient by replacing the respective coordinate in the constructed point $\mathbf{p}$ with its additive inverse.

For $\mathbf{p} = (p_1, \ldots, p_d) \in \operatorname{supp}(f)$ define the *minimal odd coordinate*

$$\operatorname{moc}(\mathbf{p}) = \min\big\{\, i \in \{1, \ldots, d\} \,\big|\, 2 \nmid p_i \,\big\},$$

where $\min \emptyset = \infty$. We use the minimal odd coordinate to partition $\operatorname{supp}^-(f) = \operatorname{supp}^{w-}(f) \,\dot{\cup}\, \operatorname{supp}^{s-}$, where

$$
\begin{aligned}
\operatorname{supp}^{w-}(f) &= \{\, \mathbf{p} \in \operatorname{supp}^- \mid \operatorname{moc}(\mathbf{p}) < \infty \,\}, \\
\operatorname{supp}^{s-}(f) &= \{\, \mathbf{p} \in \operatorname{supp}^- \mid \operatorname{moc}(\mathbf{p}) = \infty \,\}.
\end{aligned}
$$

The elements of $\operatorname{supp}^{w-}(f)$ are called *weakly negative*. They have at least one odd coordinate. The elements of $\operatorname{supp}^{s-}(f)$ are called *strongly negative*. They have exclusively even coordinates. We furthermore define $B^{w-}(f)$ and $B^{s-}(f)$ corresponding to $\operatorname{supp}^{w-}(f)$ and $\operatorname{supp}^{s-}(f)$, respectively, and we obtain

$$B(f) = B^+(f) \circ B^{w-}(f) \circ B^{s-}(f) \circ (\mathbf{0}, -1)^*.$$

Consider a matrix B obtained from $B(f)$ by deleting rows. Then we define $\operatorname{moc}(B) = \operatorname{moc}(B_{11}, \ldots, B_{1d})$, i.e., the minimal odd coordinate of $\mathbf{p} \in \operatorname{supp}(f)$ corresponding to the first row of B.

function find-positive-general(f)

    data : $f \in \mathbb{Z}[x_1, \ldots, x_d]$

    result: $\mathbf{p} \in \mathbb{Q}^d$ or `"failed"`

1    $B := B^+(f) \circ B^{w-}(f)$

2    $B^{s-} := B^{s-}(f)$

3    $\mu := \infty$

4    $h := $ `"infeasible"`

5   **while** $h = $ `"infeasible"` *and* $B \neq [\,]$ **do**

6   **if** *the first row of B is in $B^{w-}(f)$* **then**

7            $\mu := \text{moc}(B)$

8        $h := $ `lpsolve`$(B^+ \circ B \circ (\mathbf{0}, -1)^*)$

9   delete the first row from B

10   **if** $h = $ `"infeasible"` **then**

11   **return** `"failed"`

12    $(n_1, \ldots, n_d, c) := h$

13    $t := 2$

14    $(t_1 \ldots, t_d) := (t^{n_1}, \ldots, t^{n_d})$

15   **if** $\mu < \infty$ **then**

16        $t_\mu := -t_\mu$

17   **while** $f(t_1, \ldots, t_d) \leq 0$ **do**

18        $t := 2t$

19        $(t_1 \ldots, t_d) := (t^{n_1}, \ldots, t^{n_d})$

20   **if** $\mu < \infty$ **then**

21            $t_\mu := -t_\mu$

22   **return** $(t_1, \ldots, t_d)$

Algorithm 3: Function find-positive-general

After these preparations we can state our function find-positive-general in Algorithm 3. A corresponding function find-zero-general is obtained by replacing in find-zero in Algorithm 2 the call to find-positive with a call to find-positive-general. Everything else remains unchanged.

For showing the correctness of find-positive-general we are going to use the following variant of Lemma 4:

Lemma 8. Let $f \in \mathbb{Z}[\mathbf{x}] \setminus \mathbb{Z}$. Let $(\mathbf{n}, c) \in \mathbb{R}^{d+1}$ such that $N^* \cdot B(f) \cdot (\mathbf{n}, c)^T \leq -\mathbf{1}$, and let $\mu = \text{moc}(B(f)) < \infty$. Then there is $a_0 \in \mathbb{N}$ such that for all $a \in \mathbb{N}$ with $a \geq a_0$ the following holds: Define $\mathbf{t} = (t_1, \ldots, t_d) \in \mathbb{N}^d$ with $t_j = a^{n_j}$ for $j \in \{1, \ldots, d\} \setminus \{\mu\}$ and $a_\mu = -t^{n_\mu}$. Then

$$\text{sign}\big(f(\mathbf{t})\big) = -\text{sign}(f, \mathbf{p}_1).$$

Proof. We have $t_j = a^{n_j} > 0$ for $j \in \{1, \ldots, d\} \setminus \{\mu\}$, $t_\mu = -a^{n_\mu} < 0$, and $p_{1\mu}$ is odd by definition of the minimal odd coordinate. It follows that

$$0 < a^{\mathbf{n}\mathbf{p}_1} = -\mathbf{t}^{\mathbf{p}_1}. \tag{8}$$

For $i \in \{2, \ldots, s\}$ we have at least $|a^{\mathbf{n}\mathbf{p}_i}| = |\mathbf{t}^{\mathbf{p}_i}|$. This allows us to conclude

from Lemma 4 (i) that

$$|\text{coeff}(f, \mathbf{p}_1) \cdot \mathbf{t}^{\mathbf{P}_1}| > \left| \sum_{i=2}^{s} \text{coeff}(f, \mathbf{p}_i) \cdot \mathbf{t}^{\mathbf{P}_i} \right|.$$

Hence $\text{coeff}(f, \mathbf{p}_1) \cdot \mathbf{t}^{\mathbf{P}_1}$ determines the sign of $f(\mathbf{t})$. Using the inequality in (8) we obtain

$$\text{sign}\big(f(\mathbf{t})\big) = \text{sign}\big(\text{coeff}(f, \mathbf{p}_1) \cdot \mathbf{t}^{\mathbf{P}_1}\big) = -\text{sign}(f, \mathbf{p}_1). \qquad \square$$

Theorem 9 (Correctness of `find-positive-general`). *Consider $f \in \mathbb{Z}[\mathbf{x}]$. Then the following hold:*

(i) *If the function `find-positive` in Algorithm 1 does not fail on f, then `find-positive-general`(f) = `find-positive`(f).*

(ii) *The function `find-positive-general` terminates.*

(iii) *The function `find-positive-general` returns either* `"failed"` *or* $\mathbf{p} \in \mathbb{Q}^d$ *with $f(\mathbf{p}) > 0$.*

Proof. (i) The function `find-positive-general` operates on $B^+(f) \circ B^{w-}(f) \circ B^{s-}(f) \circ (\mathbf{0}, -1)^*$ while `find-positive` operates on $B^+(f) \circ B^- \circ (\mathbf{0}, -1)^*$ so that there is possibly a different order of rows lying below $B^+(f)$. However, when `find-positive` does not fail, then the same feasible solution is found in both functions before touching anything outside B^+, and in line 10 of `find-positive-general` we have exited the while-loop with $\mu = \infty$. It follows that the if-conditions in lines 15 and 20 of `find-positive-general` are always false, and the rest of the code after the while-loop is computationally equivalent to the corresponding part of `find-positive` except for an expanded notation.

Accordingly, a proof of parts (ii) and (iii) can be straightforwardly derived from the proof of (i) and (ii) of Theorem 5, respectively: If the function `find-positive` in Algorithm 1 does not fail on f, then there is nothing else to do. Otherwise we always reach lines 15 and 20 with $\mu < \infty$, replace t_μ with its additive inverse, and apply Lemma 8 instead of Lemma 4 (ii), where we know that that $\text{sign}(f, \mathbf{p}_n) < 0$. $\qquad \square$

4 Practical Issues

In this section we would like to discuss issues and share experiences related to a practical implementation of our method.

One major benefit of our approach is the reduction of an algebraic problem to linear programming (LP). Linear programming is a field with more than 50 years of active algorithmic research, strongly driven by practical applicability and aiming at robust implementations. Our own implementation combines the the Codemist Standard Lisp (CSL)-based version of the computer algebra system Reduce [16, 24, 25] with the Gurobi Optimizer [13]. Technically, CSL provides a foreign function interface that allows us to dynamically load the Gurobi C-library at runtime and call its functions from within symbolic mode Reduce functions. Gurobi uses the Simplex algorithm. So far we have got no experience

with the use of implementations of polynomial methods for LP, like the interior point method [17].

Gurobi uses floating point arithmetic with a limited precision. We want to adress some issues related to this, which we consider of general inteterst, because that floating point approach is typical for Simplex-based LP software.

On the one hand, LP solvers are quite good at controlling numerical stability. With our comprehensive computations we have never encountered any problems with false results due to LP rounding errors. On the other hand, in line 15 of Algorithm 1 we obtain $n_1, \ldots, n_d, c$ as floats with small rounding errors. These rounding errors do not affect correctness but cause a subtle problem: Converting $n_1, \ldots, n_d$ into fractions, the GCD of their denominators will typically be 1 so that the principal denominator m computed in line 18 becomes the very large product of those denominators. Consequently, in line 19 we obtain our final $\mathbf{n}$ with very large relatively prime integer coordinates. This, in turn, renders infeasible the exponentiation of increasing powers of 2 with those integer coordinates and substitution of the result into f in line 11 of Algorithm 1 or in lines 14, 17, and 19 of Algorithm 3. There are two principle ways out, which we call the *pure LP approach* and the *MIP approach*, respectively. Of course, the single design decisions made with these approaches can be recombined to yield further, mixed, approaches.

The Pure LP Approach The pure LP approach tries to get along with the delivered floats. Specifically, lines 18–20 in Algorithm 1 are skipped, and a floating point vector is returned. The while-loops in line 11 of Algorithm 1 and line 17 of Algorithm 3 remain correct with floating point exponents $\mathbf{n}$. Later, in lines 11–12 of Algorithm 2 it is important to convert to rationals. In particular the substitution of floats into a high-degree polynomial f in line 12 could cause considerable numerical instabilities. Subsequent root isolation to floating point precision in line 13 and back-substitution of the obtained floats in line 14 worked well with all our computations.

The MIP Approach MIP stands for *mixed integer (linear) programming*. Our Lemma 3 allows us to declare $n_1, \ldots, n_d$ as integers to the LP solver right away, while c remains real. As MIP is NP-hard [18], the MIP approach is considerably harder than the pure LP approach in terms of theoretical complexity. In practice there are several advanced algorithms for Simplex-based MIP solving, which rely in some way on considering an *LP relaxation*, i.e., considering integer variables as real variables, and, in the feasible case, trying to construct a mixed real integer feasible point on the basis of an LP solution. The Gurobi solver specifically uses advanced *cutting plane* [12] methods for that construction. For the largest problems discussed with our practical computations in Section 5 below, we have observed factor of about 3 for MILP solving compared to LP solving.

There is an interesting optimization with the MIP approach: Since in out situation MIP feasibility is equivalent to LP feasibility by Lemma 3, one can generally first check the latter in lines 14–15 of Algorithm 1, and in the feasible case rerun for the corresponding MIP problem. Using this strategy, there is always at most one MIP solving step per root finding problem. Furthermore, one runs MIP solving only on feasible instances. This excludes the really problematic

13   $(y_1, \ldots, y_d) = (y^{n_1}, \ldots, y^{n_d})$ for a new variable y

14   **if** $\mu < \infty$ **then**

15     $\lfloor \quad y_\mu = -y_\mu$

16   $f_1 := f(y_1, \ldots, y_d) \in \mathbb{Z}(y)$

17   $t := 2$

18   **while** $f_1(t) \leq 0$ **do**

19     $\lfloor \quad t := 2t$

20   **return** $(y_1(t), \ldots, y_d(t))$

Algorithm 4: Code to replace lines 13–22 in Algorithm 3

cases, which are LP feasible but not MIP feasible problems.

In rare cases one obtains integer solutions which are so large that they render exponentiation and substitution in line 11 of Algorithm 1 or in lines 14, 17, and 19 of Algorithm 3 infeasible. One can impose a suitable bound on the absolute values of the solutions, and in case of exceeding that bound treat the problem as infeasible, and proceed to the next candidate.

Another noteworthy optimization is the symbolic precomputation of a univariate rational function for the while-loop in line 17 of Algorithm 3. See Algorithm 4 for details. A corresponding simpler variant, of course, works also for lines 10–12 in Algorithm 1.

For root isolation in line 13 of Algorithm 2 we use the *Vincent–Collins–Akritas* method [4]. We obtain a real algebraic number encoded by a univariate defining polynomial and an open isolating interval, which is back-substituted in line 14, yielding a vector of such real algebraic numbers as the final solution **z**.

5   Some Practical Computations

We consider input polynomials originating from 4 different chemical and biological models. This yields 929 instances altogether. For all of these instances we are checking for zeros with strictly positive coordinates. It turns out that for 640 of the instances we find $B^+ = [\,]$ in line 1 of Algorithm 1, which tells us that the corresponding polynomial is positive definite (on the interior of the first hyperoctant). Running our method on the 289 remaining instances, it fails in only 7.3 percent of the cases. Table 1 shows detailed information for the single models. It also shows size (number of monomials), dimension (number of variables), and the largest degree of an occurring variable for the respective largest instance. It furthermore shows the maximal computation time for a single instance and the sum of computation times.[1] All computations have been carried out on a 2.8 GHz Xeon E5-4640 with the MIP approach, yielding exact algebraic number solutions:

Notice that for our particular application the detection of definiteness by our implementation establishes a perfect result. From that point of view, one could argue that our method fails in only 3 percent of the cases.

---

[1]All input and log files are available at `http://research-data.redlog.eu/arXiv/2015/subtropical/`.

	METH	OMBO	MBO	MAPK	Total
number of instances	7	496	405	21	**929**
number of definite instances	3	338	283	16	**640**
number of remaining instances	4	158	122	5	**289**
found zero in	4	144	107	5	**260**
failed on	0	14	15	0	**29**
failed on (% of remaining)	0	8.9	12.3	0	**7.3**
size of largest instance	347	9787	9706	863438	**863438**
dimension of largest instance	7	7	7	10	**10**
degree of largest instance	6	10	9	12	**12**
maximal time (s)	0.16	4.68	10.00	15.87	**15.87**
total time (s)	0.21	199.91	162.88	15.92	**379.92**

Table 1: Statistics for our practical computations

Acknowledgments

References

[1] A. G. Akritas and A. W. Strzebonski. A comparative study of two real root isolation methods. *Nonlinear Analysis: Modelling and Control*, 10(4):297–304, 2005.

[2] E. M. L. Beale. An alternative method for linear programming. *Mathematical Proceedings of the Cambridge Philosophical Society*, 50:513–523, 1954.

[3] F. Boulier, M. Lefranc, F. Lemaire, P.-E. Morant, and A. Ürgüplü. On proving the absence of oscillations in models of genetic circuits. In *Proceedings of the Algebraic Biology 2007*, volume 4545 of *LNCS*, pages 66–80, 2007.

[4] G. E. Collins and A. G. Akritas. Polynomial real root isolation using Descarte's rule of signs. In *Proceedings of SYMSAC '76*, pages 272–275, ACM Press, 1976.

[5] G. B. Dantzig. Linear programming and extensions. Princeton University Press, Princeton, NJ, 1963.

[6] R. A. Dwyer. On the convex hull of random points in a polytope. *Journal of Applied Probability*, 25(4):688–699, 1988.

[7] M. El Kahoui and A. Weber. Deciding Hopf bifurcations by quantifier elimination in a software-component architecture. *Journal of Symbolic Computation*, 30(2):161–179, 2000.

[8] H. Errami, M. Eiswirth, D. Grigoriev, W. M. Seiler, T. Sturm, and A. Weber. Efficient methods to compute hopf bifurcations in chemical reaction networks using reaction coordinates. In *Proceedings of the CASC 2013*, volume 8136 of *LNCS*, pages 88–99, 2013.

[9] H. Errami, W. M. Seiler, T. Sturm, and A. Weber. On Muldowney's criteria for polynomial vector fields with constraints. In *Proceedings of the CASC 2011*, volume 6885 of *LNCS*, pages 135–143, 2011.

[10] K. Gatermann, M. Eiswirth, and A. Sensse. Toric ideals and graph theory to analyze hopf bifurcations in mass action systems. *Journal of Symbolic Computation*, 40:1361–1382, 2005.

[11] K. Gatermann and S. Hosten. Computational algebra for bifurcation theory. *Journal of Symbolic Computation*, 40(4–5):1180–1207, 2005.

[12] R. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, 1963.

[13] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*, 2014.

[14] E. Hairer, S. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*, volume 8 of *Series in Computational Mathematics*. Springer, 1993.

[15] J. K. Hale and H. Kocak. *Dynamics and Bifurcations*, volume 3 of *Texts in Applied Mathematics*. Springer, 1991.

[16] A. C. Hearn and R. Schöpf. *Reduce User's Manual, Free Version*, October 2014.

[17] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[18] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, 1972.

[19] L. G. Khakhiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[20] C. E. Lemke. The dual method of solving the linear programming problem. In *Naval Research Logistics Quarterly*, volume 1, pages 36–47. 1954.

[21] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.

[22] A. Makhorin. *GNU Linear Programming Kit*. Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia, August 2014.

[23] B. Mishra. *Algorithmic Algebra*. Texts and Monographs in Computer Science. Springer, 1993.

[24] A. C. Norman. Codemist Standard Lisp (CSL) technical overview and details, July 1991.

[25] A. C. Norman. Thirty years of Lisp support for REDUCE. In *Proceedings of the A3L 2005*. BOD, Norderstedt, Germany, 2005.

[26] H. Peyrl and P. A. Parrilo. Computing sum of squares decompositions with rational coefficients. *Theor. Comput. Sci.*, 409(2):269–281, 2008.

[27] J. C. F. Sturm. Mémoire sur la résolution des équations numériques. In *Mémoires présentés par divers Savants étrangers à l'Académie royale des sciences, section Sc. math. phys.*, volume 6, pages 273–318, 1835.

[28] T. Sturm and A. Weber. Investigating generic methods to solve hopf bifurcation problems in algebraic biology. In *Proceedings of Algebraic Biology 2008*, volume 5147 of *LNCS*, pages 200–215, 2008.

[29] T. Sturm, A. Weber, E. O. Abdel-Rahman, and M. El Kahoui. Investigating algebraic and logical algorithms to solve Hopf bifurcation problems in algebraic biology. *Mathematics in Computer Science*, 2(3):493–515, 2009.

[30] B. Sturmfels. *Solving Systems of Polynomial Equations.* AMS, Providence, RI, 2002.

[31] A. J. H. Vincent. Sur la résolution des équations numériques. *Journal de Mathématiques Pures et Appliquées*, 1:341–372, 1836.

[32] D. Wang and B. Xia. Stability analysis of biological systems with real solution classification. In *Proceedings of the ISSAC 2005*, pages 354–361. ACM Press, 2005.

[33] A. Weber, T. Sturm, and E. O. Abdel-Rahman. Algorithmic global criteria for excluding oscillations. *Bull. Math. Biol.*, 73(4):899–916, 2011.