A note about *"Faster algorithms for computing Hong's bound on absolute positiveness"* by K. Mehlhorn and S. Ray

Przemysław Koprowski

Faculty of Mathematics
University of Silesia
ul. Bankowa 14
PL-40-007 Katowice, Poland

Abstract

We show that a linear-time algorithm for computing Hong's bound for positive roots of a univariate polynomial, described by K. Mehlhorn and S. Ray in an article "Faster algorithms for computing Hong's bound on absolute positiveness", is incorrect. We present a corrected version.

Key words: Polynomial roots, positive root bounds, linear time

1. Introduction

Computing an upper bound for real roots of a polynomial is an important problem in computational algebra. It has numerous applications (for instance root separation, to mention just one). Thus, in recent decades there has been intensive effort to find such bounds. Given a univariate polynomial $A = a_0 + a_1x + \cdots + a_nx^n \in \mathbb{R}[x]$ with a positive leading coefficient, a good bound for its positive roots is the one obtained in Hong (1998):

$$H(A) := 2 \cdot \max_{\substack{j<n \\ a_j<0}} \min_{\substack{i>j \\ a_i>0}} \left(\frac{-a_j}{a_i}\right)^{\frac{1}{i-j}}.$$

A naïve, straightforward implementation of this bound clearly has a time complexity of $\mathcal{O}(n^2)$. Mehlhorn and Ray (2010) proposed a geometric approach to computing this bound and presented a very smart algorithm that runs in linear time (with respect to the degree of a polynomial). Unfortunately their algorithm turns out to be incorrect. The

Email address: `pkoprowski@member.ams.org` (Przemysław Koprowski).
URL: `http://z2.math.us.edu.pl/perry/` (Przemysław Koprowski).

aim of this note is to explain the source of errors in their work as well as to propose a possible correction so that the resulting algorithm accurately computes the Hong's bound and still works in linear time.

For the sake of clarity, in this paper we adopt the notation of Mehlhorn and Ray (2010), much to a discomfort of the author, who is more used to a different notational convention. Thus, with every nonzero term $a_i x^i$ of a polynomial $A = a_0 + a_1 x + \cdots + a_n x^n$, we associate a pair $\bigl(i, -\lg|a_i|\bigr)$, where lg denotes a logarithm of base 2. We interpret these pairs as points in the real plane $\mathbb{R}^2$. A point $p = \bigl(i, -\lg|a_i|\bigr)$ is said to be *positive* if $a_i > 0$, otherwise it is called *negative*. The set of positive points with abscissæ greater or equal to some fixed integer j is denoted

$$P_j^+ := \Bigl\{\bigl(i, -\lg|a_i|\bigr) : i > j \text{ and } a_i > 0\Bigr\}.$$

The abscissæ of the points p_i for $i \in \{0, \dots, n\}$, being exponents of x in the original polynomial, are non-negative integers. Thus, if p_i is a positive point, then p_i and p_n are respectively the left-most and right-most points of the convex hull $\mathrm{CH}(P_i^+)$ of P_i^+. Consequently, they split the boundary of $\mathrm{CH}(P_i^+)$ into two chains of points: the *lower hull* $\mathrm{LH}(P_i^+)$ and the *upper hull* $\mathrm{UH}(P_i^+)$. If p is a point positioned to the left of P_i^+, then there is a unique line that passes through p and at least one point of P_i^+ and such that all points of P_i^+ are on or above it. This line is called the *lower tangent* and denoted $\tau(p, P_i^+)$. The left-most point of the intersection $\tau(p, P_i^+) \cap P_i^+$ is called the *point of tangency* of $\tau(p, P_i^+)$. The algorithm of Mehlhorn and Ray keeps track of the following data during execution:

- σ_i is the maximal slope of lower tangents computed so far;
- ℓ_i is the lower tangent to $\mathrm{CH}(P_i^+)$ with the slope σ_i;
- t_i is the point of tangency of ℓ_i;
- $\mathcal{L}_i$ is the lower hull of P_i^+.

In what follows we preserve this notation.

2. Problems

The algorithm presented in (Mehlhorn and Ray, 2010, Section 3) suffers from two interconnected problems:

Issue 1. The variable t_i—that stores the point of tangency of the lower tangent with a maximal slope—is initialized at the beginning of the algorithm (Mehlhorn and Ray, 2010, Algorithm 1, line 2) by setting $t_n := p_n$, but it is **never lowered**. Hence it stays at p_n throughout the whole process. Indeed, analyzing the algorithm, we see that:

- in line 15, t_i is being passed over by setting $t_i := t_{i+1}$;
- for $a_i > 0$, the variable t_i is not set in the pseudo-code, but according to the description (Mehlhorn and Ray, 2010, page 680, line -4) it is again set $t_i := t_{i+1}$;
- in line 10, a new t_i is searched to the **right** of t_{i+1}.

Issue 2. The claim (Mehlhorn and Ray, 2010, page 680, line -10):
"the tangent point t_i of $\ell_i = \tau(p_i, P_i^+)$ cannot lie to the left of t_{i+1}"

is false. It would be true if the lower hull of positive points has not changed since we found t_{i+1}. But not without this assumption. If the lower hull has changed, looking for the point of tangency, we must scan the entire lower hull starting from its beginning, not from the previously obtained point t_{i+1}. We show an explicit example. Take a polynomial:

$$A_\alpha := \alpha + 4x^3 - 2x^4 + 4x^5 + 8x^8, \qquad \text{with} \quad \alpha < 0.$$

For $\alpha = -1$, the polynomial has two negative coefficients and so the Hong's bound equals twice the maximum of two minimums:

$$\text{for } a_0 : \min_{\substack{i>0 \\ a_i>0}} \left(\frac{-\alpha}{a_i}\right)^{\frac{1}{i}} = \min\left\{\sqrt[3]{\frac{1}{4}}, \sqrt[5]{\frac{1}{4}}, \sqrt[8]{\frac{1}{8}}\right\} = \sqrt[3]{\frac{1}{4}},$$

$$\text{for } a_4 : \min_{\substack{i>4 \\ a_i>0}} \left(\frac{2}{a_i}\right)^{\frac{1}{i-4}} = \min\left\{\frac{1}{2}, \sqrt[4]{\frac{1}{4}}\right\} = \frac{1}{2}.$$

Thus, the maximum is reached for a pair of coefficients (a_0, a_3) and

$$H(A) = 2 \cdot \sqrt[3]{1/4} = \sqrt[3]{2} \approx 1.2599.$$

Let us analyze what the algorithm of Mehlhorn and Ray does in this case. Ignore for a moment issue 1. The coefficients of A correspond to points (see Figure 1):

$$p_1 = (0,0), \quad p_2 = (3,-2), \quad p_3 = (4,-1), \quad p_4 = (5,-2), \quad p_5 = (8,-3).$$

The lower hull $\mathcal{L}_4 = \mathrm{LH}(P_4^+)$ of $P_4^+ = \{p_4, p_5\}$ is just a line segments connecting p_4 with p_5. The lower tangent $\tau(p_3, \mathcal{L}_4)$ to the convex hull of P_4^+ and passing through p_3 is a line ℓ_3 with a slope $s_3 = \frac{(-1)-(-2)}{4-5} = -1$. The point of tangency of ℓ_4 is $t_3 = p_4$. (If we did not ignore issue 1 here, the algorithm would incorrectly pick up $t_3 = p_5$ and so s_3 would be $-1/2$, which is evidently not minimal.) The point p_3 is negative, thus the lower hull of positive points stays intact and we have $\mathcal{L}_3 = \mathcal{L}_4$.

The next point to consider is p_2. It is positive, hence the algorithm updates the lower hull. Now, $\mathcal{L}_2$ is a line segment with endpoints p_2 and p_5. The point p_4 is removed from the lower hull.

Finally, we consider the point p_1. It is negative again. The lower tangent $\tau(p_1, \mathcal{L}_2)$ is a line ℓ_1 that passes through the newly added point p_2. It has a slope of $-2/3$. It is now evident that in order to find the tangent point, we must scan the lower hull **from the beginning**, not from the point $t_3 = p_4$. Not only the point p_4 was removed from the lower hull, but scanning the points to the right of it would result in picking up a point p_5 and the resulting line would have a slope $-3/8$. All in all, the algorithm computes $\max\{-1, -3/8\} = -3/8$ and returns $H(A) = 2 \cdot 2^{-3/8} \approx 1.5422$, which is not the correct Hong's bound for A.

It should be noted that, if we take $\alpha = -8$ in the above polynomial, then the Hong's bound is obtained not from the pair (p_1, p_2) but from (p_1, p_5). This shows that if the lower hull of the set of positive points is rebuilt, we cannon a priori exclude any points. All the point of the new lower hull must be scanned to obtain the lower tangent.

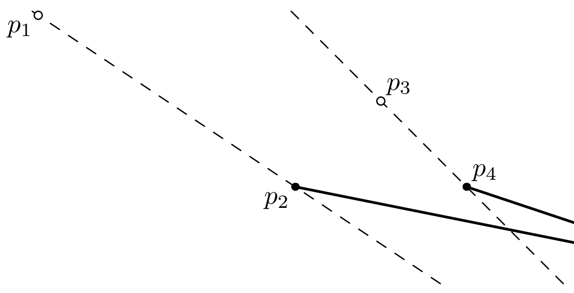

Fig. 1. Configuration of points corresponding to the polynomial $A_{-1} := -1+4x^3-2x^4+4x^5+8x^8$. Positive points are marked by filled dots, negative ones by empty dots. Thick lines correspond to lower hulls, dashed lines to lower tangents.

3. Corrections

The algorithm of Mehlhorn and Ray can be simply corrected to evade the pitfalls mention in the previous section. The most direct approach is to update the point of tangency t_i, when the lower hull of the positive points changes. A straightforward, but rather naïve, solution is to set t_i to the first point of $\mathcal{L}_i$. The corresponding part of the algorithm would then read:

- *if p_i is a positive point (i.e. $a_i > 0$), then:*
  - · *compute the lower hull $\mathcal{L}_i$ of P_i^+ as described in (Mehlhorn and Ray, 2010, Section 2);*
  - · *set $t_i := p_i$, $\sigma_i := \sigma_{i+1}$ and let ℓ_i be a line through t_i of a slope equal σ_i.*

Unfortunately, these modifications blow the time complexity of the algorithm up to $\mathcal{O}(n^2)$, which is the time complexity of the most direct evaluation of Hong's bound. An explicit example of a polynomial for which scanning all constructed lower hull from end to end would take $\mathcal{O}(n^2)$ time is the following one:

$$\left(\alpha_0+x+\alpha_1x^2+4x^3\right)+\left(\alpha_2+x+\alpha_3x^2+4x^3\right)\cdot x^4+\cdots+\left(\alpha_{2k}+x+\alpha_{2k+1}x^2+4x^3\right)\cdot x^{4k},$$

where $\alpha_0,\ldots,\alpha_{2k+1}$ are some negative real numbers. One could hope that it is enough to reset t_i only if it is one of the points that are removed, when the lower hull is rebuilt. But it is not true. The most evident example is the point $p_n = (n, -\lg a_n)$. It is an initial tangent point and is never removed from the lower hull. Nevertheless, in general when the lower hull is rebuilt, p_n will not be the tangent point any longer (the polynomial A_{-1} considered in the previous section is an example).

In order to recover the linear time complexity, we need to reshape the algorithm more seriously and separate the phase of building of the lower hulls from computation of a lower tangent with the maximal slope. The algorithm we are going to present is a two-pass process. The first pass scans the points in the decreasing order of indices (i.e. right-to-left) and is used to build and store all the lower hulls. Subsequently, in the second pass the algorithm goes through the points left-to-right and computes the sought Hong's bound.

As said, we are going to store all intermediate lower hulls $\mathrm{LH}(P_i^+)$. If we did it naïvely and store them as a list of lists, then we would end up with a space complexity of $\mathcal{O}(n^2)$. Reading and writing this data would need $\mathcal{O}(n^2)$ time. We can evade this trap, observing that if p_i is a positive point, then $\mathrm{LH}(P_i^+)$ is a chain of the form $(p_i, p_j, \text{further points})$ and its "tail" $(p_j, \text{further points})$ is a lower hull of P_j^+. Consequently, we may store all the lower hulls as a single array of indices, call it V. For a positive point p_i we set $V[i] = j$ to be the index of the **second** point (the first one is of course p_i) of the lower hull $\mathrm{LH}(P_i^+)$. Hence the lower hull of P_i^+ equals

$$\mathrm{LH}(P_i^+) = \big(p_i, p_{V[i]}, p_{V[V[i]]}, \dots, p_n\big).$$

On the other hand, for a negative point p_i, we let $V[i]$ be the index of the **first** point of $\mathrm{LH}(P_i^+)$, or in other words the index of the first positive coefficient of A to the right of a_i.

We are now ready to present a corrected algorithm that computes the Hong's bound of a polynomial in linear time.

Algorithm 1. Given a polynomial $A = a_0 + a_1x + \cdots + a_nx^n$ with a positive leading coefficient, this algorithm computes its Hong's upper bound for positive roots. In what follows, we denote $p_i := \big(i, -\lg|a_i|\big)$ for $i \in \{0, \dots, n\}$.

```
// First pass: construction of all lower hulls
(1) Set $V[n] := -1$ and initialize the index of the last visited positive point to $k := n$;
(2) iterate over the coefficients of $A$ in decreasing order of indices $i \in \{n-1, n-2, \dots, 0\}$;
    (a) if $a_i < 0$, then the lower hull $\mathrm{LH}(P_i^+)$ does not change, hence set $V[i] := k$
        and reiterate the main loop;
    (b) if $a_i > 0$, then
          (i) scan the lower hull
                $\mathcal{L}_{i+1} = \mathrm{LH}(P_{i+1}^+) = \big(p_k, p_{V[k]}, p_{V[V[k]]}, \dots, p_n\big)$
              to find the point of tangency of $\tau(p_i, P_{i+1}^+)$ as explained in Section 2 of
              Mehlhorn and Ray (2010);
         (ii) let $j \in \mathbb{N}$ be the abscissa of the point found in the previous step, set
              $V[i] := j$;
        (iii) update $k := i$;

// Second pass: computation of the Hong's bound
(3) let $j := \min\{i : a_i < 0\}$ be the lowest index of a negative coefficient of $A$;
(4) scan the lower hull $\mathrm{LH}(P_j^+) = \big(p_{V[j]}, p_{V[V[j]]}, \dots\big)$ to find the tangent point $p_k$ of
    $\tau(p_j, P_j^+)$, set $t_j := p_k$, $\ell_j := \tau(p_j, P_j^+)$ and let $\sigma_j$ be the slope of $\ell_j$;
(5) iterate over the coefficients of $A$ in an increasing order of indices starting from $j+1$:
    (a) if $a_i \geq 0$, then pass over
              $\sigma_i := \sigma_{i-1}, \quad \ell_i := \ell_{i-1} \quad \text{and} \quad \mathrm{t}_i := t_{i-1}$
        and reiterate the loop;
    (b) if $a_i < 0$ and the previous point of tangency $t_{i-1}$ lies to the left of $p_i$, then:
          (i) scan the lower hull $\mathrm{LH}(P_i^+) = (p_{V[i]}, p_{V[V[i]]}, \dots, p_n)$ to find the tangent
              point $p_m$ of $\tau(p_i, P_i^+)$;
         (ii) if the slope $s_i$ of $\tau(p_i, P_i^+)$ is greater than $\sigma_{i-1}$, then set
                $\sigma_i := s_1, \quad \ell_i := \tau(p_i, P_i^+) \quad \text{and} \quad t_i := p_m$
              otherwise pass over
                $\sigma_i := \sigma_{i-1}, \quad \ell_i := \ell_{i-1} \quad \text{and} \quad \mathrm{t}_i := t_{i-1}$
    (c) if $a_i < 0$ and the previous point of tangency $t_{i-1}$ lies to the right of $p_i$, then:
```

Table 1. Running time of a Sage implementation of a proposed linear-time algorithm (middle column) against a direct quadratic-time implementation (right column). The times are in second for 10 random polynomials of a given degree.

degree	linear	quadratic
5	0.053	0.003
10	0.127	0.005
20	0.268	0.012
50	0.711	0.038
100	1.492	0.139
200	2.979	0.473
500	7.382	2.625
1000	13.821	10.279
2000	28.116	40.378
5000	72.341	245.940
10000	148.879	991.194

(i) check if p_i lies on or above ℓ_{i-1}, if it does, the ignore p_i setting

$$\sigma_i := \sigma_{i-1}, \quad \ell_i := \ell_{i-1} \quad \text{and} \quad \mathrm{t}_i := t_{i-1}$$

(ii) otherwise, when p_i lies below ℓ_{i-1}, let k be the abscissa of t_{i-1}, scan the "tail" $(p_{V[k]}, p_{V[V[k]]}, \ldots, p_n)$ of the lower hull $\mathrm{LH}(P_i^+)$, to find the tangent point p_m of $\tau(p_i, P_i^+)$;

(iii) set σ_i to be the slope of $\tau(p_i, P_i^+)$ and update $\ell_i := \tau(p_i, P_i^+)$, $t_i := p_m$;

(6) return $H(A) = 2^{1+\sigma_n}$.

The correctness of the algorithm is immediate and its linear time complexity follows from the fact that looking for a tangent points we always scan a range of points to the right of the range previously scanned. The only points that are accessed more than once are the actual points of tangency.

Final remarks

The corrected algorithm was implemented in a computer algebra system Sage (2016), both to test it correctness and evaluate its speed. The code can be downloaded from authors home page at `http://z2.math.us.edu.pl/perry/papersen.html`. It was executed on randomly generated polynomials of varying degrees and the average computation times were compared with running times of a direct implementation of the Hong's bound. Figure 2 and Table 1 summarize the results.

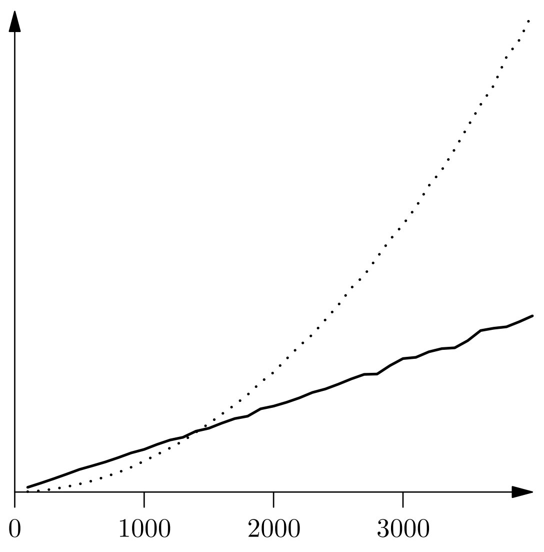

Fig. 2. Running time of a Sage implementation of a proposed algorithm (solid line) agains a direct implementation (dotted line). The horizontal axis represents degrees of random polynomials.

References

Hong, H., 1998. Bounds for absolute positiveness of multivariate polynomials. J. Symbolic Comput. 25 (5), 571–585.
URL `http://dx.doi.org/10.1006/jsco.1997.0189`

Mehlhorn, K., Ray, S., 2010. Faster algorithms for computing Hong's bound on absolute positiveness. J. Symbolic Comput. 45 (6), 677–683.
URL `http://dx.doi.org/10.1016/j.jsc.2010.02.002`

Sage, 2016. Sage Mathematics Software System (Version 7.3).
URL `\url{http://www.sagemath.org}`