New method for bounding the roots of a univariate polynomial

Eric Biagioli, Luis Peñaranda, Roberto Imbuzeiro Oliveira
IMPA – Instituto de Matemtica Pura e Aplicada
Rio de Janeiro, Brazil
w3.impa.br/∼{eric,luisp,rimfo}

***Abstract*—We present a new algorithm for computing upper bounds for the maximum positive real root of a univariate polynomial. The algorithm improves complexity and accuracy of current methods. These improvements do impact in the performance of methods for root isolation, which are the first step (and most expensive, in terms of computational effort) executed by current methods for computing the real roots of a univariate polynomial. We also validated our method experimentally.**

***Keywords*-upper bounds, positive roots, polynomial real root isolation, polynomial real root bounding.**

I. Introduction

Computing the real roots of a univariate polynomial $p(x)$ is a central problem in algebra, with applications in many fields of science and, in particular, in computer graphics. It appears, for example, in computation with curves, that are modeled using polynomials.

The first and most expensive step when computing the roots is to *isolate* them. That is, to compute a set of disjoint intervals containing the roots (exactly one root on each interval and all roots contained in some interval).

Root bounding is a central problem to real root isolation algorithms. Namely, the Vincent-Akritas-Strzeboński (VAS) algorithm and Vincent-Collins-Akritas (VCA) algorithm[1]. In particular, VAS computes a *lower bound* on the positive roots of a polynomial on each execution of the main loop. It has been shown that even slight improvements on the quality of this lower bound (and even investing much computational effort computing it) impact on the performance of the VAS algorithm [1], [2].

There are two main variables of root bounding which impact in the performance of VAS: the *accuracy of the bounds* and the *efficiency of the root bounding method*. In the present work we address both variables. We show a method which both improves the accuracy of the current methods *and* is faster.

Along the paper, we will consider the problem of obtaining upper bounds for the positive real roots of polynomials. We will introduce the concept of *killing graph* and will use it to show how all the existing approaches can be expressed as instances of it. Then, we introduce a new root bounding method, which is actually a way to *improve the result obtained by any other method*. Our method works as follows: at the beginning, an upper bound for the positive roots of the input polynomial is computed using one existing method (for example the *First Lambda* method). During the execution of this (already existent) method, our method also creates the *killing graph* associated to the solution produced by that method. It structure allows, after the method have been run, to analyze the produced output and improve it.

Moreover, the complexity of our method is $O(t + \log^2(d))$, where t is the number of monomials the input polynomial has and d is its degree. Since the other methods are either linear or quadratic in t, our method does not introduce a significant complexity overhead. To better improve the solution, it can be executed many times. We show in the paper that, with only *two* runs, our method improves all the current algorithms (even those whose complexity is quadratic in t).

Contributions: We first show all the existing methods as instances of a general concept. After that, we introduce two key observations, which allow us to *iteratively improve* the results obtained by any of the current methods.

Moreover, we implemented our method and extensively tested it, on many significant scenarios. We show that our technique yields a big improvement in the results while not introducing a significant time penalty.

II. Related Work and a General Framework

The problem of upper-bounding the biggest positive root of a univariate polynomial has been studied by many authors. We will shortly mention the most relevant current methods, and present them in an unified way. The most important methods present in the literature are: (I) Cauchy's *leading coefficient* [3] [4] [2] (II) Lagrange [4] [2] (III) Kioustelidis [5] (IV) Stefanescu [6] (V) first-λ (FL) [3] (VI) local-max (LM) [3] (VII) first-λ quadratic (FLQ) [1] [2] (VIII) local-max quadratic (LMQ) [1] [2] Methods (I), (II), (III) and (IV) are mentioned just for historical interest, methods (V) and (VI) produce better bounds with the same computational effort.

From now on we will refer to terms with positive coefficients as *positive terms* and terms with negative coefficients as *negative terms*. We will be considering the problem of giving an upper bound for the biggest positive root of a polynomial $p(x)$, whose dominant term is positive. If p had negative dominant term, we would just need to compute the bound of $-p(x)$.

[1]Currently, VAS is the default root isolation method in many widely used computer algebra systems, such as Mathematica, SageMath, SymPy and XCAS. VCA was, for many years, the default method many computer algebra systems, but now it is only used by Maple

If we have the terms $a_n x^n$, and $a_i x^i$, with $n > i, a_n > 0, a_i < 0$, we know that at some point x_0, the term $a_n x^n$ *starts* to be greater than the term $-a_i x^i$. Id est: we know that $\exists x_0 \mid x > x_0 \Rightarrow a_n x^n + a_i x^i > 0$. In fact, $x_0 = \sqrt[n-i]{a_i/a_n}$.

We say that the term $a_n x^n$ *kills* the term $a_i x^i$ with *cost* $\sqrt[n-i]{-a_i/a_n}$.

Note that if we have a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

on which (1) every negative term is killed by a higher-degree positive term and (2) there is no positive term killing more than one negative term, then the maximum of the associated costs of these killings is an upper bound for the maximum positive root of p. Let us make this observation more clear, through some examples. Consider the polynomial:

$$p_1(x) = x^5 - 25x^4 + 200x^3 - 600x^2 + 600x - 120$$

We know that

- $x^5 > 25x^4$ when $x > 25$ (x^5 kills $-25x^4$ with cost 25);
- $200x^3 > 600x^2$ when $x > 3$ ($200x^3$ kills $-600x^2$ with cost 3);
- $600x > 120$ when $x > 0.2$ ($600x$ kills $-120x^4$ with cost 0.2).

Thus, we have: (1) all negative terms are killed by higher degree positive terms and (2) there are no positive terms killing more than one negative term. Observe that $\max(25, 3, 0.2)$ is an upper bound for the maxmimum root of $p_1(x)$, since all the negative terms are killed by some positive term at this point.

Observe also that we could break positive terms in parts, and use the resulting parts separately to kill lower degree terms. For instance, if we had the polynomial

$$p_2(x) = 30x^5 - 25x^4 - 10x^3 - 200x^2 - 30$$

we could break up the term $30x^5$ into four or more parts and use four of them to kill the four terms. For instance:

$$\begin{array}{rccccccc} p_2(x) = & 1x^5 & +2x^5 & +3x^5 & +4x^5 & +18.5x^5 & +1.5x^5 \\ & -25x^4 & -10x^3 & -200x^2 & -30 \end{array}$$

And we know, for example, that • $1x^5 > 25x^4$ when $x > 25$ • $2x^5 > 10x^3$ when $x > 2.23$ • $3x^5 > 200x^2$ when $x > 4.05$ • $4x^5 > 30$ when $x > 1.49$. Thus, we have that when $x > \max(25, 2.23, 4.05, 1.49)$, $p_2(x) > 0$. Note that the way in which we broke the $30x^5$ does impact in the result we obtain.

Having stressed these two concepts, we can point out that any *strategy* of breaking positive terms and asigning positive terms (or parts of positive terms) to each one of the negative terms of a polynomial is an algorithm for computing an upper bound for the maximum positive root.

Observe that we could imagine a (directed) graph, in which the nodes are the terms of the polynomial and the edges show the relation *killed by*. Our examples would be:

$$\begin{array}{rccccccc} & \overset{25}{\swarrow}\ \searrow & & \overset{3}{\swarrow}\ \searrow & & \overset{0.2}{\swarrow}\ \searrow & \\ p_1(x) = & x^5 & -25x^4 & +200x^3 & -600x^2 & +600x & -120 \end{array}$$

$$\begin{array}{rccccccc} p_2(x) = & x^5 & +2x^5 & +3x^5 & +4x^5 & +28.5x^5 & +1.5x^5 \\ & \uparrow & \uparrow & \uparrow & \uparrow & & \\ & 25 & 2.23 & 4.05 & 1.49 & & \\ & | & | & | & | & & \\ & -25x^4 & -10x^3 & -200x^2 & -30 & & \end{array}$$

This graph, whose vertices are the monomials and whose edges are the assignments, is what we call *killing graph*. **All the current methods are, essentially, strategies to create the killing graph**. Strategies for deciding which positive term we should break and in which parts, and how to assign the arrows. We refer the reader to the bibliography of each particular method for details. Some of the methods (LMQ and FLQ) need to make a number of calculations proportional to the square of the number t of terms in the polynomials and some others (FL, LM, etc.) are linear in t. Linear methods LM and FL have been used in algebra systems like Mathematica and Sage; and have been the default method for root bounding in such systems.

First observation. Observe that we could have, for a polynomial $p(x)$, a killing graph on which the nodes are polynomials $e_1(x), e_2(x), \ldots, e_i(x)$, instead of being terms of $p(x)$. We only require that $p(x) - \sum_{1 \le j \le i} e_j(x) \ge 0$. And we could say, in a similar way, that an expression $e_1(x)$ *kills* an expression $e_2(x)$ when asymptotically $e_1(x) - e_2(x) > 0$, and that the associated cost is the biggest real zero of $e_1(x) - e_2(x)$.

Second observation. Another point that should be stressed is the following important fact. If we have a killing graph in which all the positive terms have been used, or if we have an expressions-based killing graph (like in the previous observation) in which the sum of all the expressions is exactly $p(x)$, then it occurs that *if the costs associated to the killings are all equal to some value r, then r is the maximum positive root of the polynomial $p(x)$*.

It is straightforward to see this on an example. Consider $p(x) = e_1(x) + e_2(x) + \cdots + e_i(x)$, we have that if $\forall j,\ e_j(x_0) = 0$, and $e_j(x) > 0,\ \forall x > x_0$, then x_0 is the maximum positive root of $\sum e_j$

III. Overview of the Technique

Based on the second observation, let us perform a quick analysis of our first example. Recall first the killing graph we had, and the associated costs.

$$\begin{array}{rccccccc} & \overset{25}{\swarrow}\ \searrow & & \overset{3}{\swarrow}\ \searrow & & \overset{0.2}{\swarrow}\ \searrow & \\ p_1(x) = & x^5 & -25x^4 & +200x^3 & -600x^2 & +600x & -120 \end{array}$$

Observe that we are not going to be able, by *any* method that breaks positive terms and assign arrows between positives and negatives terms, to improve the bound 25. This polynomial only accepts one unique assignment of arrows. The only thing that we could do would be to break positive terms, and we would be killing the term $-25x^4$ with *a part* of x^5 instead of using *the whole* term, so the current associated cost 25 to the killing of $-25x^4$ would become even greater, and our bound would be worst. Thus, we cannot improve the bound 25 with methods that break positive terms and assign

TABLE I
BOUNDS OBTAINED WITH DIFFERENT METHODS, FOR POLYNOMIALS OF DEGREE 100

Poly	min(FL, LM)	min(FLQ, LMQ)	min(FL-1, LM-1)	min(FL-2, LM-2)	MPR
L	$1 \times 10^{+04}$	$1 \times 10^{+04}$	6178	**4458**	375
C_I	5	5	3.916	**3.313**	0.9999
C_{II}	4.975	4.975	3.896	**3.297**	0.9995
W	5050	5050	3135	**2273**	100
M	1.041	1.041	1.036	**1.036**	1.036
RC_{20}	3.369	3.369	2.894	**2.299**	2.198
RC_{20}	1.534	1.35	1.35	**1.335**	-1.238
RC_{20}	1.553	**1.471**	1.542	1.512	1.15
RC_{1000}	100.1	100.1	99.23	**99.23**	99.22
RC_{1000}	1.454	**1.181**	1.441	1.408	1.028
RC_{1000}	2.115	1.811	1.913	**1.811**	1.199
MRC_{20}	1.63	**1.103**	1.199	1.129	0.9977
MRC_{20}	$1.185 \times 10^{+06}$	$1.185 \times 10^{+06}$	$7.898 \times 10^{+05}$	$\mathbf{5.924 \times 10^{+05}}$	$5.924 \times 10^{+05}$
MRC_{20}	$6.435 \times 10^{+05}$	$6.435 \times 10^{+05}$	$6.435 \times 10^{+05}$	$\mathbf{6.435 \times 10^{+05}}$	$6.435 \times 10^{+05}$
MRC_{1000}	$8.093 \times 10^{+299}$	$8.093 \times 10^{+299}$	$8.093 \times 10^{+299}$	$\mathbf{8.093 \times 10^{+299}}$	$8.093 \times 10^{+299}$
MRC_{1000}	$1.358 \times 10^{+301}$	$1.358 \times 10^{+301}$	$6.791 \times 10^{+300}$	$\mathbf{6.791 \times 10^{+300}}$	$6.791 \times 10^{+300}$
MRC_{1000}	$5.735 \times 10^{+300}$	$5.735 \times 10^{+300}$	$5.735 \times 10^{+300}$	$\mathbf{5.735 \times 10^{+300}}$	$5.735 \times 10^{+300}$
PoR_{20}	$8.944 \times 10^{+06}$	$6.026 \times 10^{+06}$	$6.026 \times 10^{+06}$	$\mathbf{4.616 \times 10^{+06}}$	$1.029 \times 10^{+06}$
PoR_{20}	$5.043 \times 10^{+06}$	$4.189 \times 10^{+06}$	$4.373 \times 10^{+06}$	$\mathbf{3.517 \times 10^{+06}}$	$1.039 \times 10^{+06}$
PoR_{20}	$6.679 \times 10^{+06}$	$6.679 \times 10^{+06}$	$5.949 \times 10^{+06}$	$\mathbf{4.21 \times 10^{+06}}$	$1.039 \times 10^{+06}$
PoR_{1000}	$4.443 \times 10^{+301}$	$\mathbf{2.879 \times 10^{+301}}$	$3.716 \times 10^{+301}$	$3.097 \times 10^{+301}$	$1.059 \times 10^{+301}$
PoR_{1000}	$9.365 \times 10^{+301}$	$9.365 \times 10^{+301}$	$5.9 \times 10^{+301}$	$\mathbf{4.882 \times 10^{+301}}$	$1.053 \times 10^{+301}$
PoR_{1000}	$6.419 \times 10^{+301}$	$6.419 \times 10^{+301}$	$5.787 \times 10^{+301}$	$\mathbf{4.751 \times 10^{+301}}$	$1.062 \times 10^{+301}$

MPR: maximum positive root, L: Laguerre, C_I: Chebyshev (first kind), C_{II}: Chebyshev (second kind), W: Wilkinson, M: Mignotte, RC: Random coefficients, MRC: Monic with random coefficients, PoR: Product of Roots

the killings. It does not matter how clever the method is, 25 is the best we can do. However, the maximum positive root of this polynomial is 12.64.

Let us go a bit deeper in the analysis. $x^5 > 25x^4$ when $x > 25$, but $200x^3 > 600x^2$ much earlier: it starts to be greater at $x = 3$. Then, we can just break the term $200x^3$ in two parts, and use one of these to help the x^5 in the killing of $-25x^4$. The new graph would be:

$$p_1(x) = \underbrace{x^5 \quad -25x^4 \quad +Ax^3}_{c_1} \quad +(200-A)x^3 \quad -600x^2$$

$$+600x \quad -120$$

Since our bound is the maximum of all the associated costs to the killings, in this situation our bound would be $\max(c_1, c_2, 0.2)$. Thus, the most desirable value for A would be the value that makes $c_1 = c_2$ (if it exists), and the value that makes the distance $|c_1 - c_2|$ as smaller as possible, otherwise.

In this particular case, in which all the exponents are consecutive, we could state a quadratic expression for c_1, and a linear one for c_2, both as functions of A, and obtain a cubic expression by equating $c_1(A) = c_2(A)$. In this way, we would be able to compute the best possible A, and all the associated killing costs by solving a cubic expression.

But this is a *very* particular case. What does it mean that we want $c_1 = c_2$? Recalling the second observation, the fact of $c_1 = c_2$ means that c_1 and c_2 equal the maximum positive root of $x^5 - 25x^4 + Ax^3 + (200 - A)x^3 - 600x^2 = x^5 - 25x^4 + 200x^3 - 600x^2$.

The maximum positive root of this last expression is approximately 13.4418. This value can be computed used the method of Sagraloff [7]. The complexity of doing this is $(O(t^3 \log(d\tau) \log(d))$, where t is the number of terms of the polynomial (4, in this example) , τ is the bitsize of its maximum coefficient and d is its degree.[2] Then, we have that a bound for the maximum positive root of $p_1(x)$ would be $\max(13.4418, 13.4418, 0.2)$. The bound reduced from the initial 25 to 13.4418. It is important at this point, to stress again that this improvement was not going to be feasible trough any of the current methods or trough any other method based on positive-terms-breakments and killing-assignments.

Also note that this concept can be applied more than once. In fact, what we do essentially is the following. We start with expressions. Each expression has associated cost and degree. We pick the expression with the biggest cost and the closest expression to it in terms of *difference of degree*. If there are more than one expression with the minimum distance, we pick the one with higher degree. After having picked both expressions, we compute a tight upper bound for the maximum positive root of the fewnomial obtained by adding them.[3] We join togheter the two picked expressions, and the associated costs of the resulting expression is the bound of the fewnomial. Note that if we started with a term-based killing graph, the second iteration could need to compute again the maximum positive root of a fournomial or of a 6-nomial. The third iteration could need to compute the maximum positive root of a 8-nomial and so on.

The strategy we are proposing to pick the second expression

[2]We will neglect the value of τ in the present work because we will not use the bit-complexity paradigm; this assumption yields a complexity of $O(t^3 \log^2(d))$ for this method.

[3]For this purpose, we can use the Sagraloff's method, or the method proposed by Alonso García and Galligo [8].

is arbitrary. In fact, choosing expressions that are *close* in terms of degree may not be the best possible choice. It has been done this way because *in general, when coefficients do not differ by many orders of magnitude, a part of an expression with much lower degree is not going to help significatively*. For example: if we had contributed (in previous example) to the killing of $-25x^4$ with a part of the term $+600x$ (which is killing the term -120 with a cost much lower than the cost of killing $-600x^2$ so it could seems to be the right choice at a first glance), the obtained new associated costs to the killings of $-25x^4$ and -120 by the terms x^5 and $600x$ would have been 24.97. Our improvement would have been almost insignificant. We would have improved from 25 to 24.97. Choosing this way we improved from 25 to 13.45.

IV. Implementation, Complexity and Experiments

As mentioned, our method improves the results produced by other methods. We implemented and used it to improve the bounds of the best linear methods (LM and FL, which we also implemented).

Our implementation receives as input a set of *expressions*, each one having associated a degree and a cost. It picks the expression with the maximum cost and the expression with minimum distance to it (in terms of difference of degree), choosing the one with higher degree in case of draw.

After having picked the two expressions $e_1(x)$ and $e_2(x)$, our implementation needs to compute a bound α for the maximum positive root of $E(x) = e_1(x) + e_2(x)$. This can be done using Sagraloff's method [7]. Note that, since $E(x)$ has exactly four monomials, t^3 is constant. This implies an astonishing complexity bound for the root isolation. However, we will use this fact only for the complexity analysis, because we use Mathematica [9] in our implementation to accurately isolate the biggest root of the polynomial.

After that, we remove from the input set the expressions $e_1(x)$ and $e_2(x)$, and we add the expression $E(x)$, with cost α. Our method returns the new set of expressions and associated costs. In this way, performing N iterations just consists in calling N times our method. The bound, of course, is the bigger associated cost in the set of expressions.

At this point, we can perform a complexity analysis. The first step is to run one of the LM or FL methods, which have cost $O(t)$, where t is the number of terms of the input polynomial. Then, we process the killing graph of the original assignment. Since this graph has $O(t)$ nodes and edges, the complexity of this step is $O(t)$. Finally, we run N times the cost-reduction step, which reduces to one root isolation using Sagraloff's method. The complexity of these N runs of the cost-reduction step is $O(N \log^2(d))$. Fixing $N = 2$, it turns out that the complexity of our method is $O(t + \log^2(d))$.

We have compared the results obtained with our method and with all the methods listed in section II, we executed one and two iterations of our improvement technique, above the linear methods FL and LM. The results are shown in table I.

The polynomial classes we used for the tests are commonly used in the bibliography [1], [3], [4], [2]. They are • Laguerre polynomials, • Chebyshev polynomials (first and second kind), • Wilkinson polynomials, • Mignotte polynomials, • polynomials with random coefficients of bitsize 20 and 1000, • monic polynomials with random coefficients of bitsize 20 and 1000, • polynomials which are product of random roots of bitsizes 20 and 1000

V. Results and Discussion

The obtained results exposed that the proposed method produces bounds of better quality, even when compared against more computationally-expensive methods. With only one iteration, our method improves the LMQ and FLQ bounds in almost all cases and, with two iterations, it gets even better. The few cases in which our method does not produce the best bound, it produces a bound very close to the best one. Akritas suggested [3] that taking $\min(\text{LM}, \text{FL})$ is going to produce the best bound in almost all cases. Here, based on our results, we can extend that statement by saying that taking the best of LM and FL and iterating once or twice our method is going to produce a bound with even better quality than the one expected when using FQL or LMQ (which are, of course, better than LM and FL; but, unlike our method, much more expensive).

VI. Conclusion

In this paper, we introduced a method to improve the quality of the bound for the maximum positive root of a polynomial computed by existing methods. Our technique showed to produce high quality bounds, and the complexity is linear (up to logarithmic factors). We obtained this result as a first step in trying to find possible optimizations for current real root isolation algorithms (VCA, VAS).

References

[1] A. G. Akritas, A. W. Strzeboński, and P. S. Vigklas, "Improving the performance of the continued fractions method using new bounds of positive roots," *Nonlinear Analysis: Modelling and Control*, vol. 13, no. 3, pp. 265–279, 2008.

[2] P. S. Vigklas, "Upper bounds on the value of the positive roots of polynomials," Ph.D. dissertation, University of Thessaly, 2010.

[3] A. G. Akritas, A. W. Strzeboński, and P. S. Vigklas, "Implementations of a new theorem for computing bounds for positive roots of polynomials," *Computing*, vol. 78, no. 4, pp. 355–367, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1007/s00607-006-0186-y

[4] A. G. Akritas and P. S. Vigklas, "A comparison of various methods for computing bounds for positive roots of polynomials," *Journal of Universal Computer Science*, vol. 13, no. 4, pp. 455–467, Apr. 2007. [Online]. Available: http://www.jucs.org/jucs_13_4/a_comparison_of_various

[5] J. B. Kioustelidis, "Bounds for positive roots of polynomials," *Journal of Computational and Applied Mathematics*, vol. 16, no. 2, pp. 241–244, 1986. [Online]. Available: http://dx.doi.org/10.1016/0377-0427(86)90096-8

[6] D. Ştefănescu, "New bounds for positive roots of polynomials," *Journal of Universal Computer Science*, vol. 11, no. 12, pp. 2125–2131, Dec. 2005. [Online]. Available: http://www.jucs.org/jucs_11_12/new_bounds_for_positive

[7] M. Sagraloff, "A near-optimal algorithm for computing real roots of sparse polynomials," in *Proceedings of ISSAC '14*, 2014. [Online]. Available: http://arxiv.org/abs/1401.6011

[8] M. E. Alonso García and A. Galligo, "A root isolation algorithm for sparse univariate polynomials," in *Proceedings of ISSAC '12*. New York, NY, USA: ACM, 2012, pp. 35–42. [Online]. Available: http://doi.acm.org/10.1145/2442829.2442839

[9] I. Wolfram Research, "Mathematica," Champaign, Illinios, 2015.