Implementation of Real Root Isolation Algorithms in *Mathematica*

Alkiviadis G. Akritas,
University of Kansas, Lawrence;
Alexei V. Bocharov,
Wolfram Research Inc. and Russian Academy of Science;
Adam W. Strzeboński,
Wolfram Research Inc. and Jagiellonian University, Kraków

current e-mail address: akritas@uth.gr

Abstract

In this paper we compare two real root isolation methods using Descartes' Rule of Signs: the Interval Bisection method, and the Continued Fractions method. We present some time-saving improvements to both methods. Comparing computation times we conclude that the Continued Fractions method works much faster save for the case of very many very large roots.

1 Introduction.

Isolation of real roots of univariate polynomials is the time-critical part of any algorithm for complex root isolation. Therefore the efficiency of the real root isolation is essential for developing efficient, guaranteed and precise root-finding strategies.

Present paper contains analysis of 2 different techniques for real roots isolation, both based on the Descartes' rule of signs, both proposed by the first named Author (see [3], [1], [2], see also a discussion in [4]).

We discuss the behavior of actual root isolation programs using the basic techniques along with the behaviour of a specific implementational variations thereof, using interval arithmetic for preprocessing the data.

The timing statistics over a wide variety of polynomials and segments seem to suggest that the Continued Fractions method of root isolation works significantly faster in most cases but that an actual implementation should be also able to switch to the Interval Bisection in a small class of extreme cases where the Continued Fractions appear to be dramatically inefficient.

2 Description of Algorithms

Described are two algorithms for isolation of the real roots of a squarefree rational polynomial, using Descartes' Rule of Signs.

The first, let us denote it by IBA, uses Interval Bisection method. The algorithm is described in [3], [4]. We also tried a variation of this algorithm making use of an "interval preprocessor":

IPRec(*polynomial, interval, depth, maxdepth*)

1. If *depth* is greater then *maxdepth* return *interval*.
2. Evaluate *polynomial* at *interval*.
3. If the resulting interval does not contain 0 return empty list of intervals, else return the union of
IPRec(*polynomial, left half of interval, depth+1, maxdepth*) and
IPRec(*polynomial, right half of interval, depth+1, maxdepth*).

IBA with Interval Preprocessor (IBAIP). Suppose we want to isolate the real roots of a polynomial f on an interval *[a,b]*.

1. Compute IPRec(*f, [a,b], 0, maxdepth*), where *maxdepth* is proportional to the degree of *f*.
2. Merge the adjoining intervals.
3. Compute IBA on each of the resulting intervals.

Interval Preprocessor turns out to be only a limited improvement: it helps in case of unitary polynomials (Table 1.), may help and does not hurt much for polynomials with small coefficients, but should definitely be switched off when the coefficients are big (Table 2.).

The second algorithm (ACF1978), described in [1], [2] uses the Continued Fractions method. Here is our implementation of this algorithm in *Mathematica*. The part printed in bold face is the scaling we have added to the original algorithm.

```
SACF[f_, x_] :=
  If[(f/.x → 0)==0,
    Union[posisol[Cancel[f/x],x,{1,0,0,1}][[1]],{{0,0}},
      posisol[Cancel[f/x]/.x → -x,x,{-1,0,0,1}][[1]]],
    Union[posisol[f,x,{1,0,0,1}][[1]],
      posisol[f/.x → -x,x,{-1,0,0,1}][[1]]]]

intrv[a_, b_] := If[a>b,{b,a},{a,b}]

posisol[f_, x_, {a_, b_, c_, d_}] :=
  (* f was obtained from the original polynomial  *)
  (* by the substitution  x → (ax+b)/(cx+d)       *)
  (* The function isolates the roots of the       *)
  (* original polynomial in the intrv (a/c,b/d).  *)
```

```
Module[{sols={},g,h,v,lb,a1,b1,c1,d1,sv,n,i,rn=0},
  v=variations[f,x];
  Switch[v,
    0, ,
    1, sols=If[c==0,
      {intrv[b,b+a posbound[f,x]]},
      {intrv[a/c,b/d]}],
    _, lb=poslbd[f,x];
      If[lb>1,
        h=f/.x→(lb x);
        Return[posisol[h,x,{a lb,b,c lb,d}]]];
      If[lb<1,
        h=f; a1=a; b1=b; c1=c; d1=d,
        h=Expand[f/.x→(x+lb)];
        a1=a; b1=a lb+b; c1=c; d1=c lb+d;
        If[(h/.x → 0)==0,
          h=Cancel[h/x]; rn++; sols={{b1/d1,b1/d1}}]];
      g=Expand[h/.x→(x+1)];
      If[(g/.x → 0)==0,
        sols=Append[sols,{(a1+b1)/(c1+d1),(a1+b1)/(c1+d1)}];
        g=Cancel[g/x]; rn++];
      sv=posisol[g,x,{a1,a1+b1,c1,c1+d1}];
      Switch[v-sv[[2]]-rn,
        0, sols=Union[sols,sv[[1]]],
        1, sols=Union[sols,sv[[1]],
          {intrv[b1/d1,(a1+b1)/(c1+d1)]}],
        _, n=Exponent[h,x];
          g=Expand[Dot[CoefficientList[h,x],
            Table[(1+x)^(n-i),{i,0,n}]]];
          If[(g/.x → 0)==0, g=Cancel[g/x]];
          a1=b1; b1=b1+a; c1=d1; d1=d1+c;
          sols=Union[sols,sv[[1]],
            posisol[g,x,{a1,b1,c1,d1}][[1]]]]];
  {sols,v}]
```

We use the following subprocedures:

`poslbd[f_, x_]` and `posbound[f_, x_]` give, respectively, lower and upper bound for the positive roots of f. Both bounds are powers of 2.

`variations[f_, x_]` gives the number of sign variations in the sequence of coefficients of f.

Our improved algorithm (SACF) is never slower then the original (ACF-1978), often is faster, and is much faster for polynomials with large roots (Table 3.).

In all cases except for the case of many large roots SACF is the fastest algorithm.

3 Comparison of Computation Times

All algorithms in this section are implemented in the *C* kernel of *Mathematica*, computation times are in seconds.

TABLE 1. Polynomials with randomly generated coefficients

Bit length of coeff.	Degree	IBA	IBAIP	ACF1978	SACF
10	5	0.09	0.14	0.03	0.03
10	10	0.31	0.57	0.08	0.07
10	20	2.93	3.42	0.29	0.31
10	30	8.07	8.57	1.35	1.27
10	40	19.7	20.5	0.55	0.53
10	50	20.4	18.3	0.85	0.83
100	5	0.12	0.17	0.10	0.09
100	10	0.68	1.37	0.33	0.33
100	20	4.04	4.32	0.61	0.58
100	30	6.10	6.98	0.67	0.70
100	40	5.2	24.8	1.95	1.93
100	50	34.7	52.6	6.13	6.15
1000	5	1.38	1.81	0.75	0.75
1000	10	2.75	5.18	3.28	3.28
1000	20	7.90	17.5	5.79	5.76
1000	30	48.2	71.3	24.1	24.2
1000	40	19.2	72.4	11.3	11.1
1000	50	35.5	108	14.3	14.5

Looking at this table, note, first of all interval preprocessing is not making the things better for randomly generated coefficients. Also scaling apparently haven't been applied in this class of examples, so it is ok to analyze just the IBA and ACF1978 columns.

For small coefficients (bit length 10) the advantage of the ACF1978 over IBA is quite noticeable from the very start and rapidly grows with the growth of degree up to factors of over 20 and more for high degrees.

For medium coefficients (bit length 100) the ACF1978 is at all times faster by factors of 2 to 5 .

For large random coefficients ACF is on par with IBA (actually almost everywhere slightly faster).

TABLE 2. Monic polynomials with randomly generated coefficients

Bit length of coeff.	Degree	IBA	IBAIP	ACF1978	SACF
10	5	0.31	0.36	0.028	0.028
10	10	2.26	2.28	0.072	0.061
10	20	7.11	5.69	0.28	0.27
10	30	25.7	14.8	0.89	0.89
10	40	45.4	30.7	1.73	1.78
10	50	72.8	54.4	2.56	2.61
100	5	8.17	8.22	0.11	0.09
100	10	44.5	37.4	0.23	0.23
100	20	221	222	0.83	0.83
100	30	616	601	1.03	1.03
100	40	1263	1221	2.55	2.55
100	50	2479	2318	4.47	4.32
1000	5	506	507	1.01	1.00

Presented in Table 2 are timings for polynomials with all the coefficients save the leading taken at random (the leading coefficient being 1). This table is the most serious evidence of the advantage of the ACF algorithm. The speed factor is around 30 for small coefficients and goes over hundreds for medium and large coefficients. (Note that scaling apparently did not work in this case either.)

TABLE 3. Products of factors (x-randomly generated root)

Bit length of roots	No.of roots	IBA	IBAIP	ACF1978	SACF
10	3	0.06	0.07	0.08	0.05
10	5	0.16	0.22	0.28	0.15
10	10	3.27	3.88	3.95	1.42
10	15	9.08	9.92	8.62	5.07
10	20	14.1	26.4	31.5	15.2
100	3	0.08	0.11	5.88	0.35
100	5	0.41	0.55	45.7	2.32
100	10	4.25	5.82	807	31.2
100	15	10.9	11.9	6072	84.7
100	20	30.3	43.3	???	263
1000	3	0.62	0.72	316	2.13

Table 3 contains some extreme cases of polynomials with a number of large roots. It shows the importance of scaling, which provides significant improvement for small and medium root sizes, but it also shows the necessity of switching back to IBA, when the roots are large.

TABLE 4. Chebyshev polynomials

Degree	IBA	IBAIP	ACF1978	SACF
5	0.05	0.10	0.067	0.067
10	0.81	0.90	0.37	0.37
15	2.40	2.63	1.27	1.10
20	13.0	16.1	4.98	3.98
30	49.4	230	23.3	16.5
40	152	939	64.6	39.3

Here the roots of the classical Chebyshev polynomials are isolated. This table again shows the importance of scaling, the overall advantage of the ACF method and growing disadvantage of the interval preprocessing in case of many roots.

References

[1] A.G.Akritas, "An implementation of Vincent's theorem", Numerische Mathematic 36, 1980, pp.53–62.

[2] A.G.Akritas, "Elements of Computer Algebra with Applications", John Wiley Interscience, New York, 1989.

[3] G.E.Collins, A.G.Akritas, "Polynomial real root isolation using Descartes' rule of signs", Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computations, Yorktown Heights, N.Y., pp.272–275.

[4] G.E.Collins, R.Loos, "Real zeros of polynomials", Computing Supplementum 4, 1982, pp.83–94.