SLV: a software for real root isolation

Elias Tsigaridas

SLV: a software for real root isolation

Elias TSIGARIDAS

Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA,

Laboratoire d'Informatique de Paris 6 (LIP6), Équipe POLSYS

4 place Jussieu, 75252 Paris Cedex 05, France

elias.tsigaridas@inria.fr

The problem of isolating the real roots of a univariate polynomial with integer coefficients is an important problem in computational mathematics. Given a polynomial with integer coefficients, $f = \sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$, the objective is to isolate the real roots of f, that is to compute intervals with rational endpoints that contain one and only one root of f. SLV is an open source software package written in C that provides functions for isolating the real roots of univariate polynomials with integer coefficients. It also provides functionality to approximate the isolated roots up to an arbitrary precision. Currently, it is realizes a subdivision algorithm based on Descartes' rule of sign, with modifications to improve its performance. SLV assumes that the input is a square-free polynomial. It performs all the operations using exact arithmetic based on the library gmp and it exploits as much as possible computations with dyadic numbers, that is numbers of the form $a/2^k$ where a and k are integers. It builds upon a constant memory variant of Descartes' rule of sign [3]. To minimize the number of allocations in the memory we wrote a small wrapper to use the queue.h library of OpenBSD, in order to have access to a fast implementation of lists and queues. Even though queue.h is written in C, it is a library that follows the generic programming paradigm, and if it is programmed carefully, it could be used with various data sets.

Description The source code is at http://www-polsys.lip6.fr/~elias/soft.html

The solver takes as an input a file that contains the polynomial that we want to isolate its real roots. The input file contains the degree of polynomial and then its coefficients in ascending order with respect to the degree. For example the file test.dat that comes with SLV is as follows

```
> more test.dat
```

```
5 -120 600 -600 200 -25 1
```

and corresponds to the polynomial $-120 + 600\,x - 600\,x^2 + 200\,x^3 - 25\,x^4 + x^5$.

To obtain a description of the functionality of SLV we type

```
> slv -h
```

```
The syntax is:
slv [-h] [-f file] [-i prec] [-p]
Details:
-h : prints help message
```

```
-f fname : read the coeffs from the fname
-i prec : the output intervals have width 2^(-prec)
-p : print roots (Default: no print)
```

Assuming that there is a file named `test.dat`, a sample run of the program is

```
> ./slv -f ../test.dat -i 20 -p
```

The output of SLV gives isolating intervals and various information about the isolation process.

`#bound` It is an upper bound on the magnitude of the negative and the positive real roots. If the number is b, then the roots are in the interval $(0, 2^b)$. We need this bound to "put" all the (positive) real roots inside the interval $(0, 1)$. If the bound is negative, then the real roots, if any, are already in $(0, 1)$ and so there is no need to perform a (homothetic) transformation.

`#roots` The number of negative, positive real roots, and their sum.

`#Nodes` The total number of nodes of the subdivision tree of the algorithm.

`#depth` The depth of the subdivision tree.

`#trans` The total number of Taylor shifts performed by the algorithm, i.e., $x \mapsto x + 1$.

`#homo` The total number of homothetic transformations, i.e., $x \mapsto x/2^b$, for an integer b.

`#pos_h_1` The total numbers of hacks of the first type. This hack is as follows: To obtain an estimation of the number of roots of a polynomial f in an interval $I \subset (0, 1)$, we transform f to f_I, using Möbius transformation. The roots of f in I correspond to the roots of f_I in $(0, \infty)$. Before performing the transformation, which actually corresponds to a Taylor shift, we check if all the coefficients of f are positive. Then f does not have any positive real roots.

`#pos_h_2` The total number of hacks of the second type. As in the description of `#pos_h_1` to obtain an estimation of the number of roots of a polynomial f in an interval $I \subset (0, 1)$, we transform f to f_I, using Möbius transformation. We only need f_I in order to obtain an estimation on the number of roots. To construct f_I we need to perform a Taylor shift. We construct the coefficients of f_I incrementally and we count the number of sign variations that occur. If at some point we obtain more that two sign variations, then we stop the process, as we know that we need to subdivide further the interval I.

`#half_h` The total number of the $\frac{1}{2}$-hacks. If in the interval (a, b) the polynomial admits two sign variations, then we check if the number $\frac{a+b}{2}$ isolates the two possible roots.

Experiments We perform various experiments on various data sets and we compared SLV with various available solvers. We used a linux machine having 8 cores of *Intel(R) Xeon(R) CPU E3-1275 v3 @ 3.50GHz* and 32 GB of RAM. We concentrate the experiments on polynomials with a lot of real roots. When the input consists of random polynomials, that usually have a small number of real roots, say equal to the logarithm or the square-root of the degree, then the other solvers are more efficient than SLV. This is also the case, if there are clusters of roots. We

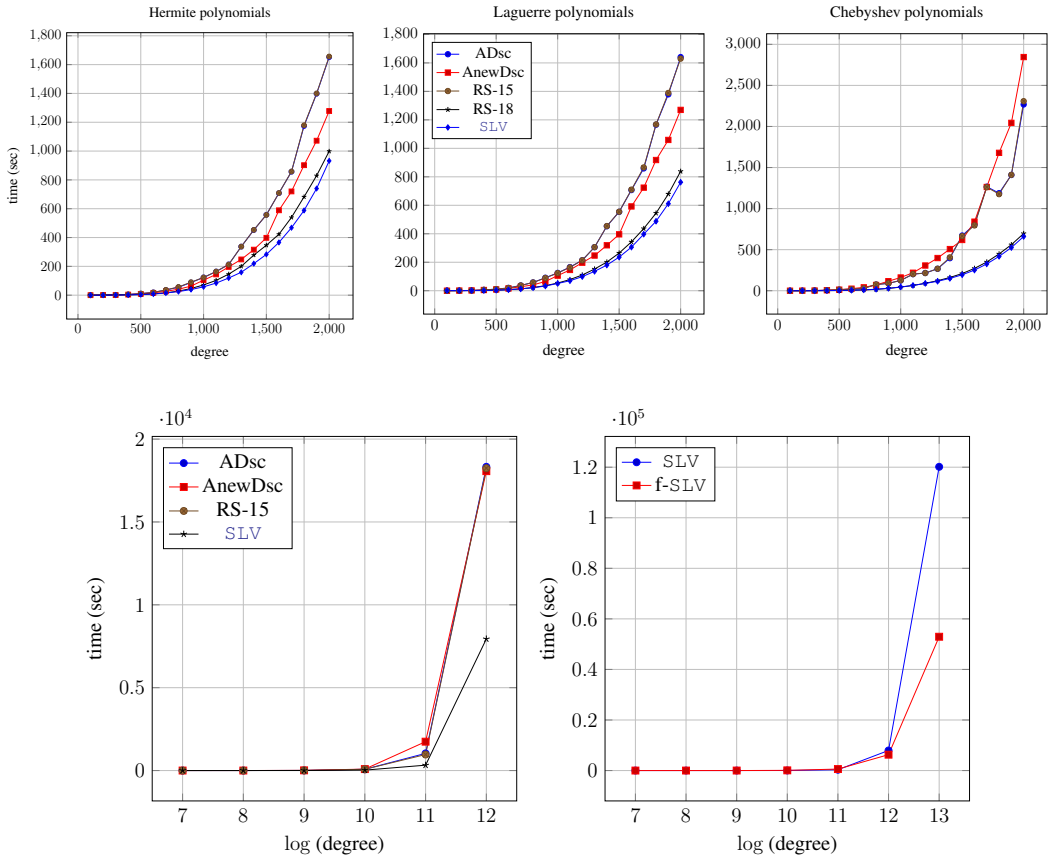

Figure 1: Graphs of timings for various experiments

refer to [1] for a detailed comparison of solvers and to [2] for recent theoretical and practical improvements for the case of clustered roots. We performed experiments with three solvers available at `http://anewdsc.mpi-inf.mpg.de`: ADsc, an implementation of approximate Descartes' algorithm, ANewDsc, an implementation of approximate Descartes' algorithm combined with Newton operator, and RS-15, a version of RS. We also tested RS-18, the version of RS in MAPLE 18. We refer to Figure 1 and Tables 1,2,3.

The next version The basic operation that `SLV` performs is the Taylor shift. The current implementation uses a suboptimal implementation of the Taylor shift that has arithmetic complexity $\widetilde{\mathcal{O}}(d^2)$. The Taylor shift is used in two cases. The first is when we need to subdivide an interval and the second when we transform an interval to $(0, \infty)$ to estimate, using Descartes' rule of sign the number of real roots in this interval. Actually, we only need to know if the number of sign variations is bigger than one. This is where the hack of the second type applies, as we construct the coefficients of the transformed polynomial incrementally. This translates to a huge gain in running times. To be able to construct the coefficients incrementally, we use the $\widetilde{\mathcal{O}}(d^2)$ algorithm. If we use an optimal algorithm, of complexity $\widetilde{\mathcal{O}}(d)$ for performing the Taylor shift, then we are not able to

construct the coefficients incrementally and the hack of the second type does not apply. To have a gain we need to combine the two algorithms so that the hack of the second type to be applicable. This is ongoing work and the implementation is in FLINT. We call this version `f-SLV`.

To give an idea of the gain in running times using this combination of algorithms and the hack of the second type we have performed experiments withe current version of `SLV` and our experimental version of `f-SLV`. We can see the results of the experiments in the right of Fig. 1 and Table 1. For $n = 13$ the experimental version `f-SLV` is more than two times faster than `SLV`.

dg	ADsc	AnewDsc	RS-15	SLV	f-SLV	#roots	bitsize
$2^7 = 128$	0	1	0	0	0	44	698
$2^8 = 256$	1	1	1	1	1	84	1486
$2^9 = 512$	7	12	7	2	5	120	3158
$2^{10} = 1024$	77	87	76	26	74	216	6626
$2^{11} = 2048$	1036	1743	966	324	586	326	14955
$2^{12} = 4096$	18328	18065	18245	7940	6282	582	31828
$2^{13} = 8192$	n/a	n/a	n/a	120155	52936	900	69784

Table 1: Timings for Katsura polynomials.

dg	ADsc	AnewDsc	RS-15	RS-18	SLV
100	0	0	0	0.1	0
200	1	1	1	0.6	0
300	2	1	2	1.8	1
400	5	4	5	2.1	2
500	11	9	11	4.2	4
600	19	16	19	8.8	8
700	36	27	37	16.3	15
800	55	41	57	28.0	25
900	88	63	90	46.3	39
1000	123	104	123	71.1	58
1100	164	144	164	102.6	84
1200	213	196	213	146.0	119
1300	337	248	337	199.9	159
1400	452	316	453	278.1	219
1500	558	398	557	346.4	283
1600	708	589	709	422.6	366
1700	857	720	858	539.6	468
1800	1174	902	1179	682.1	588
1900	1398	1072	1400	829.5	740
2000	1652	1278	1656	998.6	932

Table 2: Hermite polynomials

dg	ADsc	AnewDsc	RS-15	RS-18	SLV
100	1	0	0	0.1	1
200	0	1	1	0.6	1
300	2	2	1	1.8	1
400	6	5	6	1.8	1
500	11	9	11	3.5	4
600	20	16	19	7.2	7
700	37	28	37	13.3	13
800	58	41	58	22.5	21
900	91	65	90	35.5	33
1000	125	106	124	53.8	51
1100	166	146	164	78.5	70
1200	214	197	214	110.5	99
1300	306	247	306	152.3	136
1400	453	319	454	201.3	180
1500	554	396	555	264.4	236
1600	707	592	710	343.1	306
1700	859	724	866	436.6	397
1800	1166	918	1168	544.7	488
1900	1379	1059	1389	679.3	611
2000	1641	1270	1630	838.4	762

Table 3: Laguerre polynomials

Acknowledgments. Partially supported by HPAC (ANR-11-BS02-013) and FP7 Marie Curie Career Integration Grant.

References

[1] M. Hemmer, E. P. Tsigaridas, Z. Zafeirakopoulos, I. Z. Emiris, M. I. Karavelas, and B. Mourrain. Experimental evaluation and cross-benchmarking of univariate real solvers. In H. Kai and H. Sekigawa, editors, *Proc. 3rd ACM Int'l Work. Symbolic Numeric Computation (SNC)*, pages 45–54, New York, NY, USA, 2009. ACM.

[2] A. Kobel, F. Rouillier, and M. Sagraloff. Computing real roots of real polynomials... and now for real! In Proc. ISSAC, 2016.

[3] F. Rouillier and Z. Zimmermann. Efficient isolation of polynomial's real roots. *J. of Computational & Applied Math.*, 162(1):33–50, 2004.