BENCHMARKING SOFTWARE FOR SOLVING MULTI-POLYNOMIAL SYSTEMS

Richard V. Schmidt and Mark J. DeBonis
Department of Mathematics
Manhattan College

Abstract

Benchmarking different software which performs the same task gives students in computer science and computer engineering an opportunity to develop important skills. It also demonstrates how effective benchmarking can be applied to computer application packages designed for solving systems of multi-polynomial equations versus a new proposed method by the second author

Introduction

The second author has proposed a new way to solve multi-polynomial equations all of the same total degree in which the number of equations equals the number of unknowns. Below is a simple example of such a system with three equations in three unknowns.

$$4x^2 + 6xy + 6xz + 2y^2 + 8yz + 3z^2 = 1$$
$$2x^2 + 3xy + 7xz - 3y^2 - 3yz - 3z^2 = 0$$
$$4x^2 + 7xy + 2xz + y^2 - 7yz - 2z^2 = -4$$

Such systems are found in many applied mathematical and scientific fields. For instance, in mathematical cryptology the method of *hidden field equations* uses a multi-polynomial system for encryption in which the solution is the hidden message. Other applied examples can be found in chemistry and robotics [1] to name a few.

A trial implementation of this new method was programmed in C++ and then employing the software package Singular [2], and was shown to be feasible for small systems of equations with at most seven unknowns. This fact was made explicit by the use of a custom made benchmarking program written in Python by the first author to examine and compare both the runtime and accuracy of the new algorithm versus the computer algebraic systems Singular, CoCoA [3] and Macaulay2 [4].

Method

Solving systems of nonlinear equations is a well-known NP-complete problem, and no polynomial-time algorithm is known for solving any NP-complete problem. Some known methods for solving multi-polynomial systems employ Gröbner bases and resultants [5]. The run time for the Gröbner basis algorithm is exponential in 2^d, or doubly exponential [6] [7], where d is the number of unknowns. With resultants, the dimension of the determinant which needs to be computed in order to solve the system grows at an alarming rate. For instance, d multi-polynomial equations in d variables each of degree n yields a determinant of dimension $2^{d-1}n^{2^{d-1}-1}$ [8].

The new algorithm proposed by the second author is a generalization of a standard technique for solving systems of linear equations by multiplying by the inverse of the coefficient matrix. A set of new unknowns is introduced in the process and by solving for these new parameters we are able to solve the original multi-polynomial system. Due in part to the fact that the algorithm employs only solutions to linear systems and polynomial root finding, this method is asymptotically more efficient than existing methods.

The new proposed algorithm solves square homogeneous systems of any degree. It runs in time exponential in $d^{O(1)}$, a significant theoretical improvement. In addition, it only requires solving linear systems and finding roots of a polynomial;

There are many parallels between methods for solving linear systems and methods for solving multi-polynomial systems. Some ideas for solving multi-polynomial systems arose from generalizing techniques employed to solve linear systems. The use of Gröbner basis extends the method of Gaussian Elimination. The use of Resultants is a generalization of Cramer's Rule. The question arises whether or not one can extend the method of solving for the unknowns by multiplying by the inverse of an appropriate coefficient matrix. This new algorithm presents a way to do this. Details of this method will appear in a forthcoming paper.

The Benchmark Program

As previously stated, the algorithm presented in this paper was investigated using Big-O runtime analysis and was determined to be theoretically more efficient than the common algorithms used today. We wanted to conduct an empirical analysis to determine how well it could solve concrete problems versus other previously discussed methods.

First we compared the algorithm up against the best Computer Algebraic Systems. We determined during early testing that Matlab, Maxima and Maple took significantly longer than the prototyped algorithm implementation. Furthermore, they did not always return all available solutions for the higher-degree systems of which we were studying and in some instances simply crashed, stating it could not solve the system. Because of this, our analysis focused on competing against Singular, Macaulay2 and CoCoA, which all specialize in solving multi-polynomial systems. These applications are well recognized as the best software around by computational algebraists. Singular and CoCoA use an implementation of Gröbner Bases (an exact method), while Macaulay2 uses Polynomial Homotopy Continuation Methods (a numerical method) to solve multi-polynomial systems.

We designed a benchmarking program in Python (as well as some C++ sub-components) in order to conduct large-scale tests on the algorithm versus the previously mentioned computer algebraic systems. The benchmarking program uses a random number generator to create systems of equations as sample problems to solve. We utilize the same randomly generated equations across all software packages and performed tests to investigate runtime and accuracy. One starts by inputting the number of trials (T), and the degree of the generated polynomial system (N) into the benchmark program. The program randomly generates a solution to the polynomial system composed of N-components. We will refer to this as a known solution, which we will use later in one of our accuracy benchmarks. Next, the benchmark program randomly generates a system of square homogeneous multi-polynomial equations of degree-N such that the known solution satisfies the system. Figure I presents a flowchart for the benchmarking program.

A high-quality random number generator is needed in order to provide a good sampling of multi-polynomial systems. Quality random values are notoriously difficult to generate using deterministic algorithms. Because of this, we avoided using the C++ linear congruential random generator. Instead, we implemented the OpenSSL random bytes function, which harvests entropy from a myriad of system and hardware sources [9]. This random function is designed for the purpose of cryptographic applications, and thus will produce the best quality values possible. Our implementation is designed such that each random value is of double-precision floating-point format with a value between -5.0 and 5.0.

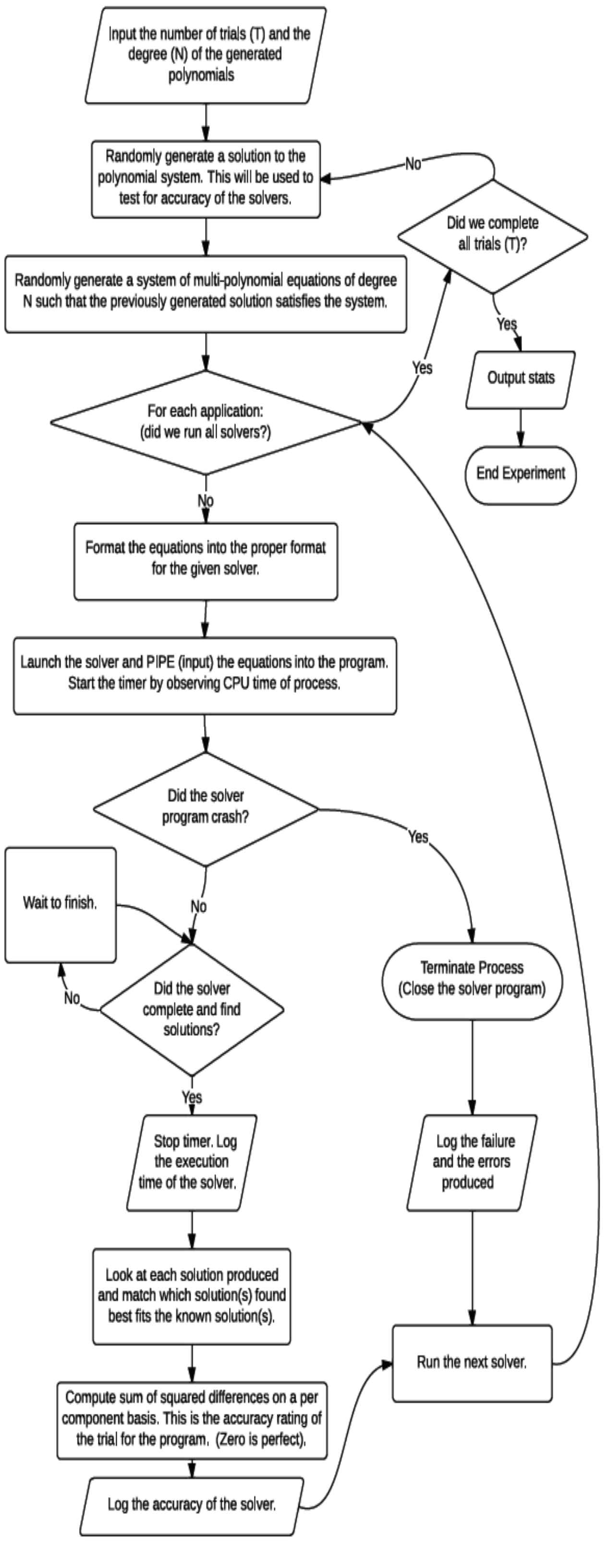
Input the number of trials (T) and the degree (N) of the generated polynomials
Randomly generate a solution to the polynomial system. This will be used to test for accuracy of the solvers.
Randomly generate a system of multi-polynomial equations of degree N such that the previously generated solution satisfies the system.
For each application: (did we run all solvers?)
No
Yes
Did we complete all trials (T)?
No
Yes
Output stats
End Experiment
Format the equations into the proper format for the given solver.
Launch the solver and PIPE (input) the equations into the program. Start the timer by observing CPU time of process.
Did the solver program crash?
Yes
No
Wait to finish.
No
Did the solver complete and find solutions?
Yes
Terminate Process (Close the solver program)
Stop timer. Log the execution time of the solver.
Log the failure and the errors produced
Look at each solution produced and match which solution(s) found best fits the known solution(s).
Compute sum of squared differences on a per component basis. This is the accuracy rating of the trial for the program. (Zero is perfect).
Log the accuracy of the solver.
Run the next solver.

Figure I: Flowchart of Benchmarking Algorithm.

It is important to note that each computer algebraic system has its own command structure and format for data inputs. Therefore, we format our created multi-polynomial system into the standardized input method corresponding to each software package. We designed the benchmark application to generate the appropriate scripts for each software package. This fully automated the testing process.

At this juncture, we are ready to assign the sample problem to a computer algebraic system. We begin by launching a new operating system process using the benchmark application. Within this process, a shell command call is made to dispatch the computer algebraic system along with its corresponding generated script to run. This will start the given computer algebraic system and immediately begin solving the problem.

Dispatching the computer algebraic system from a new system process within Python provides us with a multitude of benefits. First, it allows us to measure the amount of time it takes to solve a given problem, with a particular computer algebraic system, in seconds with 6-place decimal accuracy. We measure runtime by making calls to the Linux operating system and asking for the total CPU process time. In a shared computing environment where other applications and system tasks are running, tasks must take turns using the CPU. If our task is not running on the CPU, the timer is paused immediately by the operating system and resumed the moment our process takes control of the processor again. Thus, we are measuring how taxing the application is on the computer processor, regardless of other applications or system services running in the background. This allows us to objectively isolate other interfering processes and get a clean measurement of runtime.

Linux shells send and receive data using three standard streams of characters, among which include stdout (standard output stream), stderr (standard error stream) and stdin (standard input stream) [10].We are able to communicate between programs using a half-duplex pipe, or a one-way data channel, which redirects the standard output stream from one process to the standard input stream of another. To create a bi-directional channel, we use two pipes. Using these two pipes, we create a full-duplex (two-way) inter-program communication channel, where we can send inputs to the computer algebraic system and receive output from within our benchmark program.

There are cases where a given input can crash the computer algebraic system. When this occurs, we take the returned information from the process and we produce a log of that event for further later study. This particular runtime is not counted since the operation was not complete. For the overwhelming majority of the trials the software package completes the task. When this is the case, we take the produced solutions, log the runtime and conduct our accuracy test.

Our method of computing accuracy can be seen by considering the following simple accuracy rating example. Let A represent a known solution and B represent the best-matched solution produced by a given Computer Algebraic System. For example, suppose A = (1.0, 2.0, 3.0) and B = (0.9, 2.0, 3.1). Our relative accuracy rating can be represented as the magnitude squared of the difference of these two vectors. In our example, it would be

$$||A - B||^2 = || (0.1, 0.0, -0.1) ||^2$$

$$= (0.1)^2 + (0.0)^2 + (-0.1)^2 = 0.02$$

In this particular example, solution B would have the accuracy rating of 0.02. Note that a perfect match will produce the result of zero. Using this as a relative measurement, we are able to determine how well each software package is able to find the known solution of the multi-polynomial system. The result of the accuracy test is then logged. We repeat the process for each computer algebraic system and the implemented algorithm to complete a trial. When the trial is complete, the benchmark application will generate a new multi-polynomial system and repeat the procedure using each solver until T trials are complete.

Results

Despite the fact that all of the other software packages have had years of software engineering development, we believe that the new algorithm performs well matched against them based upon our tests with regards to five and six unknown variables.

Table I shows the results using systems of degree 6 over the course of 500 trials. We found that the new algorithm was better than CoCoA on both runtime and results. In our tests we found that CoCoA only returned 2 out of the available 64 solutions, while our algorithm returned all 64. In addition, CoCoA's runtime averaged more than three-times that of our algorithm.

With respect to accuracy, the new algorithm was better than Macaulay2 on average by a factor of ten. Note that the accuracy of our algorithm is currently limited by the polynomial root-finder. We believe the accuracy will be much closer to exact once this process has been refined. This may very well involve the use of parallel processes which our algorithm is well-suited to employ.

Table I: 500 Trials Solving Degree-6 Multi-Polynomial Systems.

Package	Average Runtime	Standard Deviation	Average Accuracy	Number Solutions
Singular	16.43 s	0.80 s	0	64
Mac2	0.87 s	0.03 s	4.40×10^{-11}	64
CoCoA	158.94 s	18.14 s	0	2
Ours	52.18 s	7.57 s	4.55×10^{-12}	64

Tables II & III summarize both runtime and accuracy results for both five and six unknowns. The reader should note the rapid increase in runtime for five versus six unknowns. For instance, for seven unknowns, Singular had a runtime of roughly an hour and for eight unknowns, Singular ran for two straight days without ever stopping.

Table II: Average runtime ± standard deviation of our solver versus the others, for d unknowns with T trials.

d	T	Ours	Singular	Mac2	CoCoA
5	687	1.47±0.19 s	0.33±0.02 s	0.50±0.03 s	0.91±0.06 s
6	500	52.18±7.57 s	16.43±0.80 s	0.87±0.03 s	158.94±18.14 s

Table III: Average accuracy of our solver versus the others, for d unknowns and T trials.

d	T	Ours	Singular	Macaulay2	CoCoA
5	687	9.36×10^{-11}	0	3.61×10^{-11}	0
6	500	4.55×10^{-12}	0	4.40×10^{-11}	0

Figure II and Figure III illustrate the results breakdown of the algorithms data with regards to 500 trials in six unknowns noted above. Figure II shows the frequency distribution of the run times of the algorithm. Notice that with the exception of two-outlier cases, all systems were solved in under 60 seconds. The mode corresponds to the average value of about 50 seconds which is most desirable. Figure III depicts the frequency distribution with regards to accuracy. The bin-interval is logarithmic based and corresponds to the exponent degree when the accuracy is written in scientific notation. Consider that nearly all of our trials had at least an accuracy comparable to Macaulay2's with a degree to -11. Most often, however we exceeded this accuracy to produce a mode degree of -12.

Conclusion

The algorithm benchmarked in this paper requires methods which run in polynomial time, namely inverting square matrices, finding roots of a polynomial and solving systems of linear equations. This approach is similar to a method called *linearization* [11] and is a standard cryptological technique for solving multi-polynomial systems (see [12] for a thorough synopsis of the various forms of this algorithm).

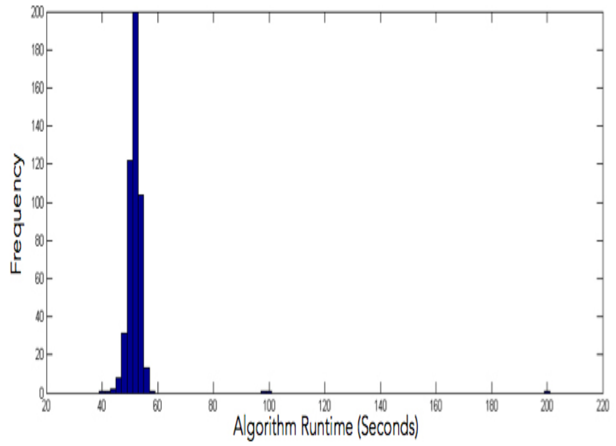

Figure II: Histogram of the runtimes.

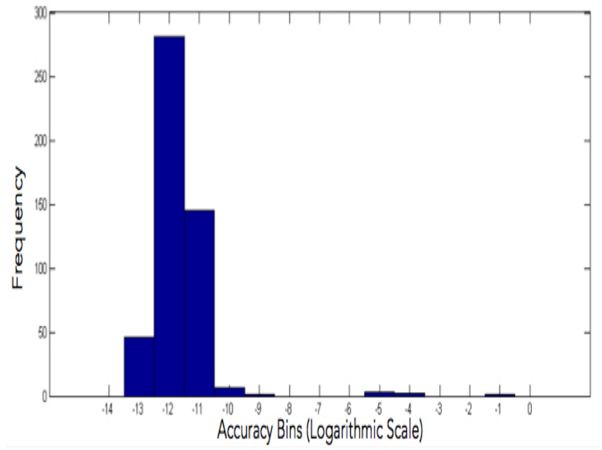

Figure III: Histogram of the log base time accuracies.

The method presented in this paper is exact except for the penultimate step in which we obtain the values of the first parameter by numerical approximation of the roots of a polynomial. Another useful feature of this algorithm is that it can be easily written to run in parallel by finding the roots of a polynomial for each unknown parameter. In the future, we hope to capitalize on the fact that our algorithm is well-suited to run in parallel in hopes of improving both runtime and accuracy.

Appendix: The Manhattan College Undergraduate Research Program

Manhattan College has a long tradition of involving undergraduates in research and was one of the original members of the Oberlin 50.

This is a group of undergraduate institutions whose students have produced many PhDs in engineering and science. At Manhattan College, students can elect to take an independent study course for three credits during the academic year. In addition, the College provides grant support to the students for ten weeks of work during the summer. Previously published articles in this journal by Manhattan College student co-authors are a very effective recruitment tool. The students have also presented their results at a variety of undergraduate research conferences including the Hudson River Undergraduate Mathematics Conference and the Spuyten Duyvil Undergraduate Mathematics Conference.

Acknowledgements

We wish to thank Manhattan College for its financial support while this research was being conducted. Mark J. DeBonis was supported by a Manhattan College faculty research summer grant while Richard V. Schmidt was supported by a Manhattan College student research summer grant.

References

1. S.D. Fox & R.H. Lewis, "Algebraic Detection of Flexibility of Polyhedral structures with Applications to Robotics and Chemistry", Fordham Undergraduate Research Journal, Vol. 2, Issue 1 (2014)

2. W. Decker, G.-M. Greurel, G. Pfister and H. Schönemann, "Singular 4-0-2 – A computer algebra system for polynomial computations", http://www.singular.uni-kl.de (2015).

3. J. Abbott, A. Bigatti, and G. Lagorio, "CoCoA-5: A System for Doing Computations in Commutative Algebra," http://cocoa.dima.unige.it (2014)

4. D.R. Grayson and M.E. Stillman, "Macaulay2, a software system for research in algebraic geometry", http://www.math .uiuc.edu/Macaulay2 (2015)

5. Cox, D., Little, J. and O'Shea, D. Using Algebraic Geometry. (Springer-Verlag. 1998).

6. J.C. Faugère, "A new efficient algorithm for computing Gröbner bases (F4)", J. of Pure and Applied Algebra, 139 (1999).

7. A. Ayad, "A Survey on the Complexity of Solving Algebraic System", International Math. Forum, **5 (**7), (2010).

8. V. Dolotin and A. Morozov, Introduction to Non-linear Algebra, (World Scientific Publishing, 2007).

9. OpenSSL Documentation Editors, "Random Numbers in OpenSSL", http://wiki.openssl. org/index.php/Random Numbers.

10. I. Shields, "Learn Linux 101: Streams, pipes, and redirects", http://www.ibm.com/ developerworks/library/l-lpic1-v3-103-4/l-lpic1-v3-103-4-pdf.pdf, (2009).

11. A. Kipnis and A. Shamir, "Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization" in H. Imai and Y. Zheng, editors, Advances in Cryptology (Crypto '99, volume 1267 of LNCS, Springer-Verlag, 1999).

12. S. Murphy and M.B. Paterson, "A Geometric View of Cryptographic Equation Solving", J. of Mathematical Cryptology, Vol. 2.

Biographical Information

Richard V. Schmidt is currently a student in the electrical and computer engineering program at Manhattan College

Mark J. DeBonis is an Assistant Professor in the Department of Mathematics at Manhattan College. He received his Ph.D. from University of California, Irvine. His research interests include computational algebraic geometry, machine learning and applied statistics.