Software for numerical algebraic geometry: a paradigm and progress towards its implementation

Daniel J. Bates* Jonathan D. Hauenstein†

Andrew J. Sommese‡ Charles W. Wampler II§

January 29, 2007

Abstract

Though numerical methods to find all the isolated solutions of nonlinear systems of multivariate polynomials go back 30 years, it is only over the last decade that numerical methods have been devised for the computation and manipulation of algebraic sets coming from polynomial systems over the complex numbers. Collectively, these algorithms and the underlying theory have come to be known as numerical algebraic geometry. Several software packages are capable of carrying out some of the operations of numerical algebraic geometry, although no one package provides all such capabilities. This paper contains an enumeration of the operations that an ideal software package in this field would allow. The current and upcoming capabilities of Bertini, the most recently released package in this field, are also described.

2000 Mathematics Subject Classification. Primary 65H10; Secondary 65H20, 65-04, 14Q99.

*Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, MN 55122 (dbates1@nd.edu, www.nd.edu/∼dbates1) This author was supported by the Duncan Chair of the University of Notre Dame, the University of Notre Dame, NSF grant DMS-0410047, the Arthur J. Schmitt Foundation, and the Institute for Mathematics and its Applications in Minneapolis (IMA)

†Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556 (jhauenst@nd.edu, www.nd.edu/∼jhauenst). This author was supported by the Duncan Chair of the University of Notre Dame, NSF grant DMS-0410047, and the Institute for Mathematics and its Applications in Minneapolis (IMA)

‡Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556 (sommese@nd.edu, www.nd.edu/∼sommese). This author was supported by the Duncan Chair of the University of Notre Dame, the University of Notre Dame, NSF grant DMS-0410047, and the Institute for Mathematics and its Applications in Minneapolis (IMA)

§General Motors Research and Development, Mail Code 480-106-359, 30500 Mound Road, Warren, MI 48090 (Charles.W.Wampler@gm.com, www.nd.edu/∼cwample1) This author was supported by NSF grant DMS-0410047 and the Institute for Mathematics and its Applications in Minneapolis (IMA)

Key words and phrases. Homotopy continuation, numerical algebraic geometry, polynomial systems, software, Bertini.

1 Introduction

Numerical algebraic geometry refers to the application of numerical methods to compute the solution sets of polynomial systems, generally over $\mathbb{C}$. In particular, basic numerical algebraic geometry embodies probability one algorithms for computing all isolated solutions of a polynomial system as well as the numerical irreducible decomposition of an algebraic set, i.e., one or more points on each irreducible component in each dimension. More recently, numerical algebraic geometry has grown to include more advanced techniques which make use of the basic methods in order to compute data of interest in both real-world applications and pure algebraic geometry.

One of the key tools used in the algorithms of numerical algebraic geometry is homotopy continuation [1, 16], a method for finding all zero-dimensional solutions of a polynomial system. Given a polynomial system $f : \mathbb{C}^N \to \mathbb{C}^n$ to be solved by homotopy continuation, one first chooses a polynomial system g which is in some way related to f but has known solutions. After forming a homotopy, such as the linear homotopy $H(x, t) = f \cdot (1 - t) + \gamma \cdot t \cdot g$, there are solution paths leading from the solutions of g to those of f which may be followed using predictor-corrector methods. Singular solutions cause numerical difficulties, so singular endgames [17, 18, 19] are typically employed. Zero-dimensional solving is discussed further in section 2.2.

The building blocks of the solution set of a set of equations are the irreducible components. These are the algebraic subsets Z of the solution set, where the set of points with neighborhoods biholomorphic to a neighborhood of a Euclidean space are connected. The solution set breaks up into a union of a finite number of irreducible components, none of which is contained in the union of the remaining components. Witness sets are the basic data structure used to numerically describe and manipulate positive dimensional solution sets. Given a multiplicity one irreducible component Z of a system of polynomials $f(x) = 0$, a witness set consists of a triple (f, L, W), where L is a random linear space of dimension $\mathrm{cod}Z$ and where $W := Z \cap L$. There is a slightly more involved definition for the case of components of multiplicity greater than one as described in [24].

Many of the algorithms of numerical algebraic geometry make abundant use of homotopy continuation. For example, the cascade algorithm [20], one of the basic methods for computing the numerical irreducible decomposition of an algebraic set, uses repeated applications of homotopy continuation at different dimensions in order to produce points on each component in each dimension. Monodromy and the trace test then lead to the complete numerical irreducible decomposition of the solution set of f. In the end, the user obtains from the methods of numerical algebraic geometry a wealth of information regarding the characteristics of the solution set of a given polynomial system, some of which may be difficult to procure by purely symbolic means. For good references on zero-dimensional solving see [13, 16] and for numerical algebraic geometry see [24].

There are several software packages available to the public which carry out some of the operations of numerical algebraic geometry. However, no one package contains all such capabilities. These packages include HOM4PS [6], PHoM [9], POLSYS [27], PHCpack [28], and HomLab [30]. The most recently released package, Bertini [2], is under ongoing development by the authors. Although all software packages were developed at different times for different reasons, they share the goal of solving polynomial systems by numerical means.

The purpose of the present paper is two-fold. One purpose is to present a paradigm for software in the field of numerical algebraic geometry. The following section contains an elaboration on the aforementioned algorithms and an enumeration of the various operations required for carrying out those algorithms, broken into four levels. Implementation-specific details, such as data structures, are omitted. The other purpose is to provide a brief introduction to Bertini and indicate its partial fulfillment of the paradigm of section 2. That is the content of section 3, which is also broken up into four levels to mirror section 2. The final section includes planned extensions of the Bertini software package.

2 A paradigm for numerical algebraic geometry software

All good software packages share several characteristics. In particular, good software should be reliable (i.e., it provides correct output with clear signals upon failure), as fast as possible with estimates of time remaining for large jobs, modular for easy modification, and user-friendly. In addition, good *numerical* software must be accurate and provide error estimates for all solutions. Reliability and accuracy generally have an adverse impact on speed, so while efficiency is important, it should not be emphasized over finding correct answers.

Numerical accuracy may be approached in two ways. One approach is to use a fixed level of precision and find as accurate a solution as possible, possibly using higher levels of precision for subsequent runs to attain more accuracy, if necessary. The other approach is to select an accuracy before the run and adjust the precision during the run (if possible) to attain that accuracy. Either way, it has recently become generally accepted that it is important to have available multiple levels of precision when implementing numerical routines.

The purpose of this section is to provide an enumerated paradigm for software specifically in the field of numerical algebraic geometry. This detailed list is broken into four levels, beginning with very basic operations not specific to polynomial system solving at level 0 in section 2.1 and moving through extensions of basic numerical algebraic geometry at level 3 in section 2.4. The operations of each level build upon the capabilities of the previous level. Each of the following four sections begins with a discussion of the necessary operations of the given level and the resulting capabilities of the software. Each section then concludes with a brief list of the main operations to be implemented at that level. All operations should be implemented for various levels of precision, ideally for arbitrarily high precision.

2.1 Level 0: Basic operations

At the very core of a numerical polynomial solver, one must of course have access to basic arithmetic both of complex numbers and complex matrices. It is important to optimize the efficiency of this arithmetic as much as possible, particularly in high precision, as most operations in numerical algebraic geometry rely heavily upon arithmetic. In addition to basic matrix arithmetic, standard techniques from numerical linear algebra are needed. Among the most important are Gaussian elimination for linear solving, QR factorization for least squares and orthogonal complements, and the SVD for finding numerical ranks. See [5, 26] for general references on numerical linear algebra, while [14] provides a more efficient method for determining the numerical rank of a matrix.

Random numbers play a key role in numerical algebraic geometry as many statements hold generically, i.e., for almost all random choices, thereby making the resulting algorithms hold with probability one. Any standard random number generator will suffice, although it is best to have the chosen random complex numbers of approximately unit modulus, for stability. It is also important to have a consistent mechanism for extending the precision of a randomly chosen complex number. In particular, upon extending the precision of a random number α to make $\hat{\alpha}$, truncating back to lower precision, and then again extending the precision, one should once again obtain $\hat{\alpha}$.

It is of course necessary to somehow obtain the polynomials of interest from the user, although the specific procedure for doing so is implementation-specific. To build a general solver, it is important to allow the functions to be defined as expressions built from subexpressions. This is beneficial not only for ease of use, but also for efficiency and numerical stability. It is also necessary for generality to allow for homotopies that depend on parameters, including analytic expressions as discussed in more detail in the following section. If the user is specifying the entire homotopy, it is also necessary to have a way for the user to provide solutions to the start system g. Otherwise, the automatically generated start system should be solved by the software.

Regardless of how the input data is provided, parsed, and stored, it is at times necessary for the software to automatically homogenize the polynomials provided by the user. Homogenization is simply the mechanism for moving from a product of complex spaces (perhaps one) to a product of complex projective spaces (perhaps one). This is a purely symbolic operation which should be implemented in such a way as to be easily reversed in case the need arises. Once a system has been homogenized, v random nonhomogeneous linear equations should be appended to the system in order to choose a patch on the product of complex projective spaces, where the projective space is given as a product of v projective spaces. These v linear equations are known as the patch polynomials.

Basic path tracking makes heavy use of both function evaluation and Jacobian evaluation. Basic function evaluation is straightforward, although it could be optimized via specialized techniques such as Horner's method. The Jacobian (i.e., the matrix of partial derivatives of the polynomials) should be computed automatically by some form of automatic differentiation so that the user does not need to provide it as input. It is generally believed that the Jacobian should be computed explicitly rather than approximated numerically, for stability.

Some of the operations at this level are common to numerical software in general, regardless of the specific application. It should be noted that existing libraries, such as LAPACK, provide robust implementations of some of these operations, albeit in one level of precision.

Summary of level 0 operations:

- Complex scalar and matrix arithmetic
- Matrix operations (e.g., linear solving, numerical ranks, and orthogonal complements)
- Random number generation
- Parsing of input data
- Function homogenization
- Function evaluation
- Jacobian evaluation
- All numerical operations should be available in multiprecision

2.2 Level 1: Basic tracking and zero-dimensional solving

It is now possible to glue together the capabilities of level 0 to build algorithms leading up to a full zero-dimensional solver, i.e., a method for computing all isolated solutions of f. The concept of "solving" a system could be interpreted two ways. Given a desired accuracy $\epsilon \in \mathbb{R}^+$, let $S := \{s \in \mathbb{C}^N : |f(s)| \leq \epsilon\}$. This set may break up into k disconnected pieces, say $S = S_1 \cup \ldots \cup S_k$. Then one interpretation of solving is to find a $z \in S_i$ for each i. The second way to interpret "solving" f is to find, for each $s \in \mathbb{C}^N$ such that $f(s) = 0$ and s has multiplicity m as a root of f, a set $\{z_1, ..., z_m\} \subset \mathbb{C}^N$ such that $|z_i - s| \leq \epsilon$ for all i. The solutions in the former sense of solving depend upon the scaling of the polynomials while those of the latter depend upon the scaling of the variable. The latter is more widely accepted as correct and is thus the definition used throughout this paper.

As discussed in section 1, homotopy continuation casts the system f in a family of polynomial systems, another of which is g. Such parameterized families sometimes arise naturally and are of great interest in some applications. Other families are artificially constructed for the specific purpose of solving f by continuation, in a so-called *ab initio* homotopy. Naturally, parameterized families can be used to define homotopies that often have fewer paths to follow than an *ab initio* homotopy. As a result, software in numerical algebraic geometry should address both *ab initio* and natural parameter homotopies. The natural parameter spaces may be complex analytic and thus the need for evaluating complex analytic expressions of parameters and complex numbers as discussed in the previous section.

At its core, homotopy continuation is simply a sequence of steps advancing the path variable and updating the solution vector x to match. Each step forward consists of a predictor step followed by a sequence of corrector steps. The prediction is carried out by Euler's method, which steps along the tangent to the path, is generally regarded as an acceptable method for this, although simple secant prediction could also be employed, as could higher order methods such as Runge-Kutta. Once a prediction has been made to a new value of t, it is necessary to refine the point back towards the solution curve. This may be accomplished by fixing t and running a Newton-like corrector until the norm of the Newton residual has dropped below a prespecified tolerance.

Naturally, there are times when steps will fail, for instance, due to insufficient precision. Such failures need not trigger path failure. Rather, adaptive steplength should be utilized. Upon step failure when using adaptive steplength, the steplength is decreased by a prespecified factor in the hope that convergence will occur for a smaller step. If progress along the path becomes excessively slow, the path is declared a failure. Conversely, if the steps are progressing well, it is worthwhile to try to increase the steplength. More details regarding prediction, correction, and adaptive steplength may be found in [1, 16, 24].

Path failure may occur for a number of reasons, but the presence of a singularity, particularly at $t = 0$, is a common cause. There are several sophisticated algorithms known as endgames that help to speed convergence at $t = 0$ for both nonsingular and singular endpoints. These endgames are typically employed for every path, so it is important to have at least one implemented in any software package for numerical algebraic geometry. Details regarding endgames may be found in [17, 18, 19].

For zero-dimensional solving, polynomial systems given by the user could be nonsquare with $n > N$. Fortunately, Bertini's theorem [24] guarantees that a new system consisting of N generic linear combinations of the original N polynomials will have among its solutions all the isolated solutions of the original system, though possibly with increased multiplicity. It may also have nonsingular isolated extraneous solutions. The extraneous solutions are easily detected as they do not satisfy the original system.

Unless the user chooses to specify a parameter homotopy, the software must be able to automatically produce an appropriate start system g, solve it, and attach it to the homogeneous, square system f in order to create the homotopy H. There are several known ways of producing start systems, although the general rule is that the computational cost increases in order to produce start systems with fewer paths to track. Among the common choices, total degree start systems, consisting of polynomials of the form $x_i^{d_i} - 1$, where d_i is the degree of the i^{th} polynomial in f, are trivial to build and solve but have the largest number of paths to be tracked (that being the product of the degrees of the functions). At the other end of the spectrum, the construction and solution of sophisticated polyhedral homotopies involve far more computation time but may result in far fewer paths (the number of which is the mixed volume). It is not clear *a priori* which type of start system is best-suited for an arbitrary polynomial system, so it is important to have multiple types of start systems available.

Once all (or most) of the aforementioned operations have been implemented, it is possible to compute the zero-dimensional solutions of a given polynomial system. Two other useful tools belong at this level. First, deflation [11] is a symbolic means of constructing a new polynomial system $\hat{f}$ from f such that $\hat{f}$ has a nonsingular solution in place of a particular singular solution of f. This helps to speed the convergence of Newton's method if run at $t = 0$. The major drawback of implementing deflation is that decisions need to be made about the rank of the Jacobian matrix. These decisions require the use of some sort of endgame which defeats the purpose of speeding up Newton's method. The big advantage of deflation is dealing with multiple components of positive dimension.

The other useful tool at this level is a post-processor to manipulate and display the solutions computed by the solver as well as any statistics gathered during tracking. As the functionality of such a tool is application-specific, no more details will be discussed here.

Summary of level 1 operations:

- Differential equation solving, e.g., Euler's method
- Newton's method
- Basic path tracking with adaptive steplength control
- Adaptive precision path tracking
- Endgames
- Squaring of systems
- Start system and homotopy generation
- Start system solving
- Full zero-dimensional solving

- Deflation
- Post-processing of zero-dimensional data

2.3 Level 2: Positive-dimensional solving

The solution set Z of a polynomial system f may have several components and these may not all have the same dimension. Letting $D := \dim Z$, the irreducible decomposition may be written as $Z = \cup_{i=0}^{D} Z_i = \cup_{i=0}^{D} \cup_{j \in \mathbb{I}_i} Z_{i,j}$, where each $Z_{i,j}$ is an irreducible component of dimension i, and accordingly each Z_i is the pure i-dimensional component of Z.

One of the key objectives in numerical algebraic geometry is to find a numerical irreducible decomposition of Z, which consists of sets $Z_{i,j} \cap L_{N-i}$, where L_{N-i} is a generic linear subspace of dimension $N-i$. $W_{i,j}$, together with L_{N-i}, is known as a witness set for $Z_{i,j}$, and by abuse of notation, $W_i = \cup_{j \in \mathbb{I}_i} W_{i,j}$ is called a witness set for Z_i. We briefly describe the algorithms for computing the irreducible decomposition below. Full details may be found in the references cited below or in [24].

The main steps in computing a numerical irreducible decomposition are:

- find witness supersets $\hat{W}_i \supset W_i$ for each dimension i,
- prune out "junk points" from $\hat{W}_i$ to extract the witness sets W_i, and
- break W_i into distinct sets $W_{i,j}$, the witness sets for the irreducible components $Z_{i,j}$.

The witness supersets $\hat{W}_i$ are generated by the application of zero-dimensional solving to find the isolated points in the slice $Z \cap L_{N-i}$. All of the $\hat{W}_i$, for $0 \leq i \leq D$, can be obtained using the cascade algorithm [20], starting at $i = D$ and cascading sequentially down to $i = 0$. The junk points in $\hat{W}_i$ must lie in Z_j, $j > i$. Thus, the junk may be removed by testing each point $p \in \hat{W}_i$ for membership in a higher-dimensional component. This can be done using continuation on slices to see if any of the witness sets W_j, $j > i$ connect to p as the slicing linear space L_{N-i} is moved continuously until it contains p. The final break up of W_i into irreducibles is accomplished by first using monodromy, which comes down to discovering connections between witness points as the linear slicing space is moved around a closed loop in the associated Grassmannian [21]. This is followed by checking if the connected groups so discovered are complete by a trace test [22]. The trace method may also be used to complete a partial decomposition. Both monodromy and the trace method involve specialized continuation of slices and careful bookkeeping.

Squaring is a concern for positive-dimensional solving just as it is for zero-dimensional solving. Although much carries over, one difference is that the size to which the system should be squared depends on the dimension of the component, e.g., for components of dimension k, the defining system should be randomized to a system of $N - k$ equations.

Given witness data for an algebraic set Z, there are three operations of particular interest for users. First, a user might want to find many points on a specific component. This is known as sampling and is very closely related to monodromy. Second, a user might want to know if a given point lies on an irreducible component of Z. This is component membership and is nearly identical to junk removal. Finally, a user might want the accuracy of some endpoint sharpened. This is just a matter of running Newton's method appropriately, perhaps after deflating f. Of course, as in the previous section, a post-processor would be appropriate, although the exact functionality is again application-specific.

Deflation "comes into its own" as a method for multiple components, e.g., to track intersections of a multiplicity greater than one component with a one-parameter family of linear spaces of complementary dimension or to sample a multiple component. Roughly speaking, if Z is a k-dimensional component of the solution set of a system $f = 0$, then the computation of the deflation at $Z \cap L$ for f restricted to an codimension k linear space gives rise to a "deflation" of the whole component. At the expense of increasing the number of variables, this allows us to numerically work with only multiplicity one components.

Summary of level 2 operations:

- Continuation of slices
- Monodromy
- Traces
- Squaring of systems for positive-dimensional solving
- Cascade algorithm
- Full numerical irreducible decomposition
- Sampling
- Component membership
- Endpoint sharpening
- Post-processing of witness data
- Deflation for components

2.4 Level 3: Extensions and applications of the basics

This highest level consists of operations that make use of the basic numerical algebraic geometry maneuvers described in the previous three levels. For example, for two algebraic sets X and Y which are the solution sets of polynomial systems f and g, respectively, suppose one has witness data W_X and W_Y for X and Y but would like a witness set W for $X \cap Y$. There is a now a method [23] for computing the numerical irreducible decomposition of such an intersection, the inclusion of which, while not essential for basic software in numerical algebraic geometry, will be important as the field continues to develop.

There are several other advanced algorithms that should be included in a complete state-of-the-art implementation. Another such technique is that of [25], which provides a way of finding exceptional sets, i.e., the sets of points in the parameter space above which the fiber has dimension higher than the generic fiber dimension. Fiber products play a key role in this algorithm and therefore need to be available in the software before the method for finding exceptional sets may be implemented. Also, in real-world applications, real solutions are often of more interest than complex solutions, so the extraction of real solutions from the numerical irreducible decomposition, for example, by an extension of the method of [15], would be very useful.

This is not intended to be a complete list, and it is anticipated that many more operations could be added in the near future. One capability, though, that is important now and will only become more essential over time, is parallelization. Although not every algorithm of numerical algebraic geometry is fully parallelizable, basic path tracking is easily parallelized so great savings can be made throughout levels 1, 2, and 3 by doing so [10, 12, 27, 29].

Summary of level 3 operations:

- Intersection of components
- Fiber products
- Finding exceptional sets
- Extracting real solution sets from complex witness
- Parallelization

3 Bertini

Bertini [2] is a new software package under ongoing development by the authors for computation in the field of numerical algebraic geometry. Bertini itself has evolved from a program called Polysolve created by Bates, C. Monico (Texas Tech University), Sommese and Wampler, although nothing substantial remains in Bertini from Polysolve.

Bertini is written in the C programming language and makes use of several specialized libraries, particularly lex and yacc for parsing and GMP and MPFR for multiple precision support. The beta version of Bertini was released in October 2006 to coincide with the Software for Algebraic Geometry workshop of the Institute for Mathematics and its Applications. It is currently anticipated that Bertini 1.0 will be made available to the public sometime in 2007. Bertini is currently only available as an executable file for 32- or 64-bit Linux and for Windows, via Cygwin. Specific instructions for using Bertini are included with the distribution and on the Bertini website.

Among other things, Bertini is capable of producing all complex isolated solutions of a given polynomial and witness sets for each positive-dimensional irreducible component. The goal of the Bertini development team is to eventually include in Bertini all operations described above in section 2. The purpose of this section is to indicate briefly which of those operations are already available in the beta version of Bertini. The specific algorithms implemented are also described, when appropriate. Details regarding the development plans for Bertini 1.0 and beyond may be found in sections 4.1 and 4.2, respectively.

3.1 Level 0

By default, Bertini uses IEEE double precision, although it also allows for any fixed level of precision available in MPFR. In particular, precision is available starting from 64 bits, increasing in 32 bit increments. Furthermore, the beta version of Bertini allows the user to select adaptive precision for zero-dimensional solving. In adaptive precision mode, Bertini begins in double precision and increases precision as necessary, determined by the algorithm described in [3].

All matrix operations described in section 2.1 have already been implemented in Bertini with the exception of the QR algorithm, which will be implemented in the next release. Gaussian elimination with scaled partial pivoting is used for linear solving. The SVD algorithm is based on the description provided in [26].

Bertini uses the basic random number generator found in the C standard library. Each random number is scaled to be of the appropriate modulus and stored as an exact rational number with the denominator a power of ten. Thus, to increase precision, all new digits are set to zero, although this is not ideal. It would be better to increase precision using randomly chosen digits such that subsequent truncations and extensions result in the same additional digits. This will be changed in a future version of Bertini.

The user provides polynomials (either the entire homotopy or just the target system) to Bertini in a file which is parsed using lex and yacc. The polynomials are stored in straight-line format for easy automatic differentiation and homogenization as well as efficient evaluation. Any optimization, such as using Horner's method, is the responsibility of the user. Homogenization is carried out as the polynomials are being parsed, and differentiation is carried out after the entire system has been parsed, currently via forward automatic differentiation as described in [8].

Bertini allows the user to include the path variable, parameters, constants, and subfunctions when defining the target system or homotopy. There are various restrictions placed on the homogeneous structures allowed depending upon the scenario, but multiple variable groups are generally supported. All coefficients, constants, and other numbers are stored as exact rational numbers, as described for random numbers above, and are assumed to be exact. As a result, function and Jacobian evaluation are available at any level of precision. Analytic functions of constants and parameters are not yet supported.

3.2 Level 1

All level 1 operations of section 2.2 have been implemented in Bertini with the exception of deflation. Euler's method is used as the predictor, and Newton's method as the corrector. A fractional power series endgame [19] is available in basic precision, fixed higher precision, and adaptive precision.

All polynomial systems are automatically squared to the appropriate number of polynomials and variables, although the details of the squaring mechanism are excluded from the present paper for brevity. If the user specifies only the target system and variable groupings, Bertini multihomogenizes the system according to the groupings and attaches a compatible m-homogeneous start system via a linear homotopy. Each polynomial in the start system consists of a product of random linear functions matching the homogeneous structure of the corresponding target polynomial. If there is only one variable group, the result is a total degree start system. Bertini also has a post-processor which collects the endpoint data for the run and creates a number of files, some of which are human-readable and some of which are better suited for machine reading. The files provide lists of singular endpoints, nonsingular endpoints, real endpoints, and so on. Important path characteristics, such as an estimate of the error in the endpoint, are also supplied in these files. Bertini also indicates whether any paths failed (and why they did) and whether any path crossing occurred prior to the endgame.

3.3 Level 2

All of the operations of level 2 described in section 2.3 have been implemented in double precision in Bertini, with the exceptions of endpoint sharpening and deflation. Before a positive-dimensional run, users have the option of computing the numerical irreducible decomposition of the polynomial system input file, sampling a component of a system for which witness data has previously been computed, or performing component membership on such a system. The algorithms are then carried out as described in section 2.3 and [24].

In the case of computing the numerical irreducible decomposition, Bertini stores all data in memory until the end of the run. At the end of the run, a postprocessor sorts the data and creates a number of files, as in the case of a zero-dimensional run. The structure of the algebraic set is also provided in a table on the screen, namely the number of irreducible components of each degree in each dimension. Additional details may be found in the Bertini documentation available in [2].

3.4 Level 3

None of the level 3 operations described in section 2.4 are available in the beta version of Bertini. Much of the further development of Bertini will involve level 3 operations.

4 The future for Bertini

It is of course impossible to predict what will happen with either Bertini or numerical algebraic geometry in the distant future. However, there is a short-term development plan for Bertini. The following sections describe the development plan for Bertini 1.0 and further planned developments beyond Bertini 1.0.

4.1 Bertini 1.0

There are several key developments currently in the process of being implemented in order to move Bertini from the beta version to version 1.0. The four main improvements are the inclusion of the cascade algorithm in fixed multiple precision as well as adaptive precision, deflation in both the zero- and positive-dimensional cases, endpoint sharpening (as described in section 2.2) and the QR algorithm. Other improvements are minor, such as several small changes to the output files and an improved post-processor. The goal is to have most of the operations of levels 0, 1, and 2 available in Bertini 1.0 in each fixed level of precision and adaptive precision.

4.2 Beyond Bertini 1.0

While there will certainly be a number of changes to the core functions of Bertini after version 1.0 is released (such as improving the way that random numbers are handled), many of the major changes will involve the implementation of the operations at level 3 and new algorithms currently being developed.

Other planned extensions include the extension of Bertini to allow analytic functions of parameters and constants, the inclusion of polyhedral methods, e.g., using MixedVol [7], and various forms of parallelization. Bertini currently operates by setting up files with input data, calling Bertini, and reading the results from files. It is anticipated that Bertini will eventually include a scripting language or make use of software that provides an interactive environment. Bertini will also undergo a move from the C programming language to C++ sometime after the release of version 1.0, both for increased modularity and for easier extension. Finally, Bertini will eventually make use of more efficient multiprecision numerical libraries as they become available and will include interfaces to other software packages commonly used in the mathematics community.

References

[1] E.L. Allgower and K. Georg. *Introduction to numerical continuation methods. Classics in Applied Mathematics, 45.* SIAM, Philadelphia, 2003. Reprint of the 1990 edition (Springer-Verlag, Berlin).

[2] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini: Software for Numerical Algebraic Geometry. Available at www.nd.edu/∼sommese/bertini.

[3] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Multiprecision path tracking. Preprint, 2007.

[4] D.F. Davidenko. On a new method of numerical solution of systems of nonlinear equations. *Doklady Acad. Nauk SSSR (N.S.)* 88:601–602, 1953.

[5] J.W. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[6] T. Gao and T.Y. Li. HOM4PS. Software available at www.csulb.edu/∼tgao.

[7] T. Gao, T.Y. Li, and M. Wu. Algorithm 846: MixedVol: A software package for mixed-volume computation. ACM Trans. Math Software, 31(4):555–560, 2005. Software available at www.csulb.edu/∼tgao.

[8] A. Griewank. *Evaluating derivatives. Principles and techniques of algorithmic differentiation. Frontiers in Applied Mathematics, 19.* SIAM, Philadelphia, 2000.

[9] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani. PHoM—a polyhedral homotopy continuation method for polynomial systems. *Computing* 73(1):55–77, 2004. Software available at www.is.titech.ac.jp/∼kojima.

[10] T. Gunji, S. Kim, K. Fujisawa, and M. Kojima. PHoMpara—parallel implemenation of the Polyhedral Homotopy continuation Method for polynomial systems. *Computing* 77(4):387–411, 2006.

[11] A. Leykin, J. Verschelde, and A. Zhao. Evaluation of jacobian matrices for Newton's method with deflation for isolated singularities of polynomial systems. In *SNC 2005 Proceedings. International Workshop on Symbolic-Numeric Computation.* Xi'an, China, July 19-21, 2005. Edited by Dongming Wang and Lihong Zhi. pp. 19-28.

[12] A. Leykin, J. Verschelde and Y. Zhuang. Parallel Homotopy Algorithms to Solve Polynomial Systems. *Mathematical Software - ICMS 2006, Second International Congress on Mathematical Software, Castro Urdiales, Spain, September 1-3, 2006, Proceedings.* Lecture Notes in Computer Science, Springer, 4151: 225-234, 2006.

[13] T.Y. Li. Numerical solution of multivariate polynomial systems by homotopy continuation methods. *Acta Numer.* 6:399–436, 1997.

[14] T.Y. Li and Z. Zeng. A rank-revealing method with updating, downdating, and applications. *SIAM J. Matrix Anal. Appl.* 26(4):918–946, 2005.

[15] Y. Lu, D.J. Bates, A.J. Sommese, and C.W. Wampler. Finding all real points of a complex curve. Preprint, 2006.

[16] A.P. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[17] A.P. Morgan, A.J. Sommese, and C.W. Wampler. Computing singular solutions to nonlinear analytic systems. *Numer. Math.* 58(7):669–684, 1991.

[18] A.P. Morgan, A.J. Sommese, and C.W. Wampler. Computing singular solutions to polynomial systems. *Adv. Appl. Math.* 13(3):305–327, 1992.

[19] A.P. Morgan, A.J. Sommese, and C.W. Wampler. A power series method for computing singular solutions to nonlinear analytic systems. *Numer. Math.* 63(3):391–409, 1992.

[20] A.J. Sommese and J. Verschelde. Numerical homotopies to compute generic points on positive dimensional algebraic sets. *J. Complexity* 16(3):572–602, 2000.

[21] A.J. Sommese, J. Verschelde, and C.W. Wampler. Using monodromy to decompose solution sets of polynomial systems into irreducible components. *Applications of algebraic geometry to coding theory, physics and computation(Eilat, 2001), NATO Sci. Ser. II Math. Phys. Chem., 36*, Kluwer Acad. Publ., Dordrecht, 2001.

[22] A.J. Sommese, J. Verschelde, and C.W. Wampler. Symmetric functions applied to decomposing solution sets of polynomial systems. *SIAM J. Num. Anal.* 40(6):2026–2046, 2002.

[23] A.J. Sommese, J. Verschelde, and C.W. Wampler. Homotopies for intersecting solution components of polynomial systems. *SIAM J. Num. Anal.* 42(4):1552–1571, 2004.

[24] A.J. Sommese and C.W. Wampler. *Numerical solution of systems of polynomials arising in engineering and science.* World Scientific, Singapore, 2005.

[25] A.J. Sommese and C.W. Wampler. Exceptional sets and fiber products. Preprint, 2006.

[26] G.W. Stewart. *Matrix algorithms. Vol. I. Basic decompositions.* SIAM, Philadelphia, 1998.

[27] H.-J. Su, J. McCarthy, M. Sosonkina, and L.T. Watson. Algorithm 857: POLSYS_GLP – A parallel general linear product homotopy code for solving polynomial systems of equations. *ACM Trans. Math. Software* 32(4): 561–579, 2006. Software available at www.vrac.iastate.edu/∼haijunsu.

[28] J. Verschelde. Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Software* 25(2): 251–276, 1999. Software available at www.math.uic.edu/∼jan.

[29] J. Verschelde and Y. Zhuang. Parallel implementation of the polyhedral homotopy method. *ICPPW '06: Proceedings of the 2006 International Conference Workshops on Parallel Processing.* IEEE Computer Society, Washington, DC, 481–488, 2006.

[30] C.W. Wampler. HomLab: Homotopy Continuation Lab. Software available at www.nd.edu/∼cwample1.