PHCmaple: A Maple Interface to the Numerical Homotopy Algorithms in PHCpack[*]

Anton Leykin[†] Jan Verschelde[‡]

10 June 2004

Abstract

Our Maple package *PHCmaple* provides a convenient interface to the functions of *PHCpack*, a collection of numeric algorithms for solving polynomial systems using polynomial homotopy continuation, which was recently extended with facilities to deal with positive dimensional solution sets. The interface illustrates the benefits of linking computer algebra with numerical software. *PHCmaple* serves as a first step in a larger project to integrate a numerical solver in a computer algebra system.

1 Introduction

PHCmaple is an interface to `phc` (**p**olynomial **h**omotopy **c**ontinuation), a program to solve polynomial systems, which is part of *PHCpack* [19].

Solving polynomial systems is a fundamental problem which occurs frequently in various models of science and engineering. Homotopy continuation methods (see e.g. [1], [8], and [9]) are a class of reliable and efficient symbolic-numeric solvers. Continuation methods are numerical as they repeatedly apply Newton's method to achieve global convergence. A homotopy method creates a family of polynomial systems (i.e.: the homotopy), linking the system to be solved with a so-called start system whose solutions are known, or easier to find. For efficiency, homotopy methods must exploit the structure of the polynomials in the system and consider the polynomials not merely as functions as continuation methods do. So we may regard the application of a homotopy method as a symbolic operation.

The package *PHCpack* implements various homotopy techniques to approximate all isolated complex roots of a polynomial system. The first release of the source code is documented by [19]. In [16] is a description of recent extensions to *PHCpack* to deal with positive dimensional solution sets. *PHCpack* contains the source code to build the program `phc`, available for various platforms: Sun workstations running Solaris, MacOS X computers, PCs under Linux and Windows.

Maple itself does, of course, not need an introduction. In most realistic applications, computer algebra is needed to derive and formulate the polynomial equations. And equally important is the analysis and visualization of the results obtained by a solver. So the interaction with computer algebra is not only natural, but also vital in the solving process. A simple Maple procedure,

[*]This material is based upon work supported by the National Science Foundation under Grant No. 0105739 and Grant No. 0134611.

[†]Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA. *Email:* leykin@math.uic.edu, *URL:* http://www.math.uic.edu/~leykin.

[‡]Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA. *Email:* jan@math.uic.edu, *URL:* http://www.math.uic.edu/~jan.

operating as a shell around the blackbox solver of *PHCpack*, was first presented in [16]. In this paper, we describe a more elaborate interface, implemented using a module which allows to use this interface like a Maple package. *PHCmaple* provides format conversions to bring systems and solutions from Maple into `phc` and from `phc` into Maple. The interface gives access to the main functions implemented in *PHCpack*.

Besides a useful application of computer algebra, an important research direction in recent years has been the application of numerical methods to deal with problems of symbolic computation, like the approximate GCD for instance, for more, see [18]. One important task in dealing with positive dimensional solution sets involves the decomposition into irreducible components. For one polynomial in several variables, this task specializes to the so-called absolution factorization, which has received a lot of attention in computer algebra lately. Our package *PHCmaple* is a modest step towards the practical integration of a numerical solver in a computer algebra system.

This paper is organized in two parts. We first present the facilities in *PHCmaple* to deal with approximate complex isolated roots before listing the functions to deal with positive dimensional solution sets.

2 Isolated Solutions of Polynomial Systems

The core of *PHCpack* contains a powerful robust engine for computing approximations to all complex isolated solutions of a square (i.e.: as many equations as unknowns) polynomial system. *PHCmaple* provides access to the blackbox solver of *PHCpack*, in addition to the root refiner and path tracker. One nice feature of *PHCmaple* is the procedure `drawPaths` which visualizes the homotopy continuation method.

As a running example, consider the following system of two equations in two variables:

$$\left\{ \begin{array}{l} x^2 + y^2 - 1 = 0 \\ x^3 + y^3 - 1 = 0. \end{array} \right. \tag{1}$$

It has four solutions:

$$(1,0),\ (0,1),\ (-1 + i\frac{\sqrt{2}}{2}, -1 - i\frac{\sqrt{2}}{2}),\ \text{and}\ (-1 - i\frac{\sqrt{2}}{2}, -1 + i\frac{\sqrt{2}}{2}), \quad \text{where} \quad i = \sqrt{-1}. \tag{2}$$

These are the exact solutions. With *PHCmaple* we will obtain numerical approximations for the solutions of (1). First we must load the `phc.module` and define the location of the executable version on our system, as done on the first line below:

```
> with(phc): setPHCloc("C:\\CODE\\MAPLE\\phc"):
> T := makeSystem([x,y], [], [x^2+y^2-1, y^3+x^3-1]):
```

The command in the previous line assigned to `T` the system in (1). In the following subsections, we show how to solve the system and refine the solutions to arbitrary precision. Then we give an example of the path tracker and illustrate the facility to draw paths.

2.1 phc[solve]

The simplest (to use) function of *PHCpack* is the blackbox solver, which, given a square polynomial system, returns a list of approximations for the complex isolated roots of the system. No other input is required from the user. The blackbox solver uses default values for the tolerances and parameters in the continuation and does not request the assistance of the user in the selection of the start system in the homotopy – see [19] for the details of the blackbox solver in *PHCpack*.

Below, we show how the corresponding *PHCmaple* function `solve` is called for our example.

```
>  sols := solve(T): printSolutions(T,sols):

(1) [x = -1.0-.707106781186548*I, y = -1.0+.707106781186548*I]

(2) [x = -1.0+.707106781186547*I, y = -1.0-.707106781186548*I]

(3) [x = .279457787737754e-6+.217714179751182e-6*I, y =
1.0+.542516561071245e-19*I]

(4) [x = 1.0+.361738783176125e-20*I, y =
.340643163100248e-6+.779553989101843e-8*I]

(5) [x = 1.0+.190718110222908e-18*I, y =
-.48510524311408e-6+.21712658445583e-6*I]

(6) [x = -.430219446274724e-6+.401671773723477e-7*I, y =
1.0+.296514987452363e-19*I]
```

Every solution returned by phc comes with three attributes: err is the magnitude of the last correction term of Newton's method; rco is an estimate for the inverse condition number of the Jacobian matrix at the root; and res is the norm of the solution evaluated at the system. For example, for the first solution of the list sols, these three attributes can be retrieved as follows:

```
>  sols[1]:-err; sols[1]:-rco; sols[1]:-res;
```

$$0.1570\,10^{-15}$$
$$0.1921$$
$$0.8882\,10^{-15}$$

To see the err attribute for all solutions, we may proceed as follows:

```
>  map(s->s:-err,sols);
```

$$[0.1161\,10^{-15},\ 0.1161\,10^{-15},\ 0.4846\,10^{-6},\ 0.7098\,10^{-6},\ 0.5797\,10^{-6},\ 0.6599\,10^{-6}]$$

From the list of err attributes we see that the last four ones are not as small as the first two ones. An explanation for this can be seen from the estimates for the inverse condition numbers:

```
>  map(s->s:-rco,sols);
```

$$[0.1921,\ 0.1921,\ 0.2295\,10^{-6},\ 0.1539\,10^{-6},\ 0.2295\,10^{-6},\ 0.2171\,10^{-6}]$$

While the first two solutions have low estimates for the condition numbers, the four last condition numbers indicate that the last six decimal places of the last four solutions may be erroneous. Looking at the list of six solutions more closely, we see that the last four solutions appear in two pairs of two close solutions, close to $(1, 0)$ and $(0, 1)$ – compare to the sequence of exact solutions in (2).

2.2 phc[refine]

We can make the radius of a cluster of approximate roots around a singularity as small as we like by applying Newton's method to the list of solutions. Among the options passed to refine the user can specify the number of computed decimal places, error tolerance, residual tolerance, the tolerance for singular solutions and the maximal number of iterations in the Newton method.

```
>  refsols :=
>  refine(sols,sys,digits=32,residual_tol=1e-30,error_tol=1e-30,singular_
>  tol=1e-20,max_iterations=30):
>  printSolutions(sys,refsols);

(1) [x = -1.0-.70710678118654752440084436210487*I, y =
-1.0+.70710678118654752440084436210487*I]

(2) [x =
-.99999999999999999999999999999999+.70710678118654752440084436210484*I
, y = -1.0-.70710678118654752440084436210486*I]
```

```
(3) [x =
.5205306546258975757408672784475e-15+.40552410972977444517883039027289
e-15*I, y = 1.0+.3505927241699136526923965746685e-45*I]

(4) [x = 1.0+.2338568649677147242777207000165e-46*I, y =
.63431698580416056902491862863394e-15+.1452444158759210076552664246326
2e-16*I]

(5) [x = 1.0+.1552688703302692381783873561629e-45*I, y =
-.45016560336118588358467402009776e-15+.202939621089520760569377491202
32e-15*I]

(6) [x =
-.80134639254142622225381788624168e-15+.748172410079813982038502708816
46e-16*I, y = 1.0+.1916189751539815163261542576703e-45*I]
```

We see that the condition numbers of the last four solutions become worse as we approach the true singularities, as the `err` attributes become smaller.

```
>  map(s->s:-rco,refsols);
```

$$[0.1921,\ 0.1921,\ 0.4274\,10^{-15},\ 0.2866\,10^{-15},\ 0.2135\,10^{-15},\ 0.4043\,10^{-15}]$$

```
>  map(s->s:-err,refsols);
```

$$[0.3142\,10^{-30},\ 0.3735\,10^{-30},\ 0.9260\,10^{-15},\ 0.6492\,10^{-15},\ 0.6555\,10^{-15},\ 0.8761\,10^{-15}]$$

As we see from the result of the `refine` that $(1, 0)$ and $(0, 1)$ are each approached by two lll-conditioned solutions, we can conclude that $(1, 0)$ and $(0, 1)$ occur each with multiplicity two. So counted with multiplicity, system (1) has six roots.

2.3 phc[track]

One may use `track` to control what start system is used for the continuation. The function takes the start and the target systems as parameters as well as a, possibly partial, list of solutions to the start system. It tracks the paths starting at these solutions outputting the corresponding solutions of the target system.

In our example we take the start system where each equation of the target system is matched with the equation of the same degree in one variable, such that its solutions are the roots of unity. The total number of solutions to such a system is exactly the Bézout number for the target system.

```
>  S := makeSystem([x,y], [], expand((1+I)*[x^2-1, y^3-1])):
```

The `1+I` multiplication factor in the start system is used to avoid singular points along the solution path defined by the homotopy. Normally, we would choose a complex number which is somewhat more random than `1+I`.

```
>  rt := evalf(-0.5+sqrt(3)/2*I): # principal cubic root of 1
>  startSolutions := [makeSolution([1,1]), makeSolution([-1,1]),
>  makeSolution([1,rt]),makeSolution([-1,rt]),
>  makeSolution([1,rt^2]), makeSolution([-1,rt^2])]:
>  L := track(T,S,startSolutions,20):
```

If the optional last parameter is specified and is equal to n, then `track` subdivides the continuation paths in n parts, thus computing $n - 1$ intermediate points on each of them. We will use the `L` from above in `DrawPaths` below.

2.4 phc[drawPaths]

The output of `track` can be used to sketch the continuation paths via `drawPaths`. For each variable, this function makes plots of the projections of continuation paths onto the corresponding complex plane.

```
>  R := drawPaths(L,S):
>  plots[display](R[1]);
>  plots[display](R[2]);
```

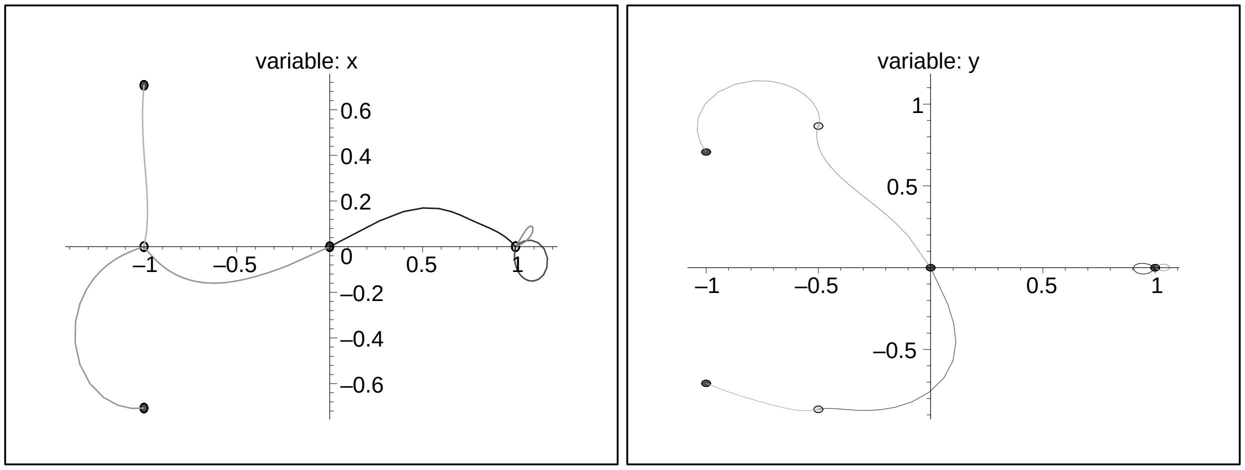

Figure 1: The output of `drawPaths`. The light disks indicate the start solutions, while the target solutions are marked by dark disks.

3 Positive Dimensional Solution Sets

We represent a k-dimensional solution set of a system by adding k random hyperplanes to the system. The isolated solutions of this augmented system are generic points and their cardinality is as many as the degree of the solution set. This augmented system with these generic points forms a so-called *witness set* for the k-dimensional solution set. A witness set is the basic data structure in a numerical irreducible decomposition [12].

A running example for this section is the system of three equations in three variables:

$$\left\{ \begin{array}{c} x^2 + y^2 + (z-1)^2 - 1 = 0 \\ (z-0.5)(z-1)y = 0 \\ (z-0.5)(z-1)x = 0. \end{array} \right. \tag{3}$$

Its solution set (see Figure 2 for the picture in the real space) consists of two isolated points, which are the intersection of the sphere and the line $x = y = 0$, as well as two one-dimensional irreducible curves: the two circles produced by slicing the sphere with the planes $z = 1$ and $z = 0.5$.

3.1 phc[embed]

The case of the positive dimension is reduced to the zero dimensional case by embedding the system, i.e.: by adding as many random hyperplanes as the dimension of the solution set. We can always obtain a square system by the addition of so-called *slack* variables. This embedding was proposed in [11].

The function `embed` creates an embedded system, which is one of the ingredients in a witness set for a k-dimensional solution set. Besides the original polynomial system, it is the responsibility of the user to provide `embed` with the expected top dimension of its solution set as the second parameter.

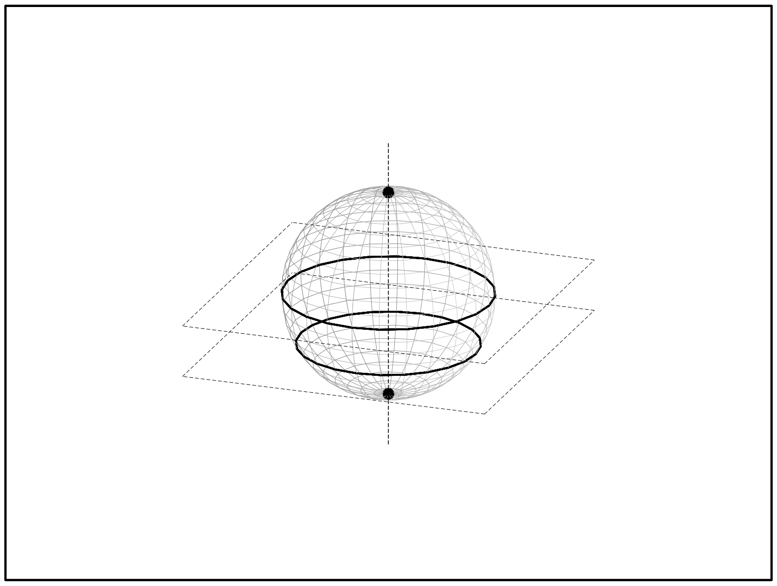

Figure 2: The zero set of the system (3) consists of two circles and two isolated points.

```
>  sys := makeSystem([x,y,z],[],[x^2+y^2+(z-1)^2-1,
>  (z-0.5)*(z-1)*y,(z-0.5)*(z-1)*x]):
>  embsys := embed(sys,1):
>  embsys :-vars;
```

$$[x, y, z]$$

```
>  embsys :-slacks;
```

$$[zz1]$$

```
>  printSystem(embsys):

(1) x^2+y^2+z^2-2*z+.722061869022232*zz1-.691828488358293*I*zz1

(2)
y*z^2-1.50000000000000*y*z+.500000000000000*y+.676798806780202*zz1-.73
6168034582387*I*zz1

(3)
x*z^2-1.50000000000000*x*z+.500000000000000*x+.388610077788065*zz1+.92
1402304881833*I*zz1

(4)
.639493055710925*x+.768796872845815*I*x+.331649046286345*y-.9434028355
35996*I*y-.126120769421903*z-.992014894807748*I*z+.768367525344066*zz1
-.640008863998488*I*zz1-.802750399628069+.596315181675744*I
```

3.2 phc[cascade]

We can now run `solve` to compute the solutions to the embedded system, which together with the embedding form a witness set for the top dimensional component of the solution set of the original system.

```
>  embsols := solve(embsys):
```

The function `cascade` launches the so-called *cascade of homotopies* in order to compute witness sets for the solution components in every dimension. This cascade was first presented in [11].

```
> comps := cascade(embsys,embsols):
> dim0 := comps[1]:  dim1 :=comps[2]:
```

The result is a list of witness sets representing the equidimensional decomposition of the variety. In our example we have two components: of dimensions 0 and 1.

You may see that we obtained 6 points in dimension 0:

```
> printSolutions(dim0:-system,dim0:-points);

(1) [x = 0, y = 0, z = 2.0]

(2) [x = 0, y = 0, z = 0]

(3) [x = -.923027707251813+.593279148760967*I, y =
.923027707251813+.593279148760967*I, z = 1.0+.481482486096809e-34*I]

(4) [x = -.799365442876975+.513794814362604*I, y =
.799365442876975+.513794814362604*I, z = .50-.137297740176043e-33*I]

(5) [x = .923027707251812-.593279148760966*I, y =
-.923027707251812-.593279148760966*I, z = 1.0+.188079096131566e-36*I]

(6) [x = .799365442876975-.513794814362604*I, y =
-.799365442876976-.513794814362604*I, z = .50+.700649232162409e-45*I]
```

However, there are clearly only two isolated solutions! Where do the other four come from?

3.3 phc[filter]

The points we find at the end of the paths in the cascade form a witness superset. Some points may belong to higher dimensional components and not to the current component in the cascade. Therefore, we have to filter out these junk points. If we know that the current component has multiplicity one, then this filtering can be done by a mere inspection of the condition number at the end of the paths, as the points on higher dimensional components are singular, and thus ill conditioned. The filtering facility `filter` calls the routine of *PHCpack* that uses the so-called homotopy membership test, as derived in [13].

```
> dim0f := filter(dim1, dim0):
> printSolutions(dim0f:-system,dim0f:-points);

(1) [x = 0, y = 0, z = 2.0]

(2) [x = 0, y = 0, z = 0]
```

3.4 phc[decompose]

Once we have a witness set for all positive dimensional solutions sets of a polynomial system, we are interested in the irreducible decomposition. For every irreducible component, we would like one witness set. The decomposition of a witness set is predicted by monodromy [14] and certified by the linear trace [15].

Below we use *PHCmaple*'s `decompose` to find the irreducible one-dimensional components of degree two.

```
> printSolutions(dim1:-system,dim1:-points);

(1) [x = -.685502648680706-1.0493407160131*I, y =
-1.2991446273126+.5536918870099*I, z = .50+.464208177178903e-16*I,
zz1 = .283451800337563e-16+.423353079656887e-16*I]
```

```
(2) [x = -1.01896486399796-1.33092346330283*I, y =
-1.57349962005932+.86187770780964*I, z =
1.0-.906089000592713e-16*I, zz1 =
.801744344776168e-16+.104102228345155e-15*I]

(3) [x = 1.24838627747489-.59841477216379*I, y =
.808428382967238+.924080358318904*I, z =
1.0+.259880445936573e-15*I, zz1 =
.130604196067113e-15+.135595403619785e-15*I]

(4) [x = 1.04530104586469-.375339338250658*I, y =
.551560327114017+.711332167200034*I, z =
.50-.130819268578456e-15*I, zz1 =
.525404099686002e-16+.517037529448153e-16*I]
```

```
> irr1 := decompose(dim1): nops(irr1);
```

$$2$$

These are, indeed, the two circles, each represented with a 2-point witness set.

```
> printSolutions(irr1[1]:-system,irr1[1]:-points);
```

```
(1) [x = -.685502648680706-1.0493407160131*I, y =
-1.2991446273126+.5536918870099*I, z = .50+.464208177178903e-16*I,
zz1 = .283451800337563e-16+.423353079656887e-16*I]

(2) [x = 1.04530104586469-.375339338250658*I, y =
.551560327114017+.711332167200034*I, z =
.50-.130819268578456e-15*I, zz1 =
.525404099686002e-16+.517037529448153e-16*I]
```

```
> printSolutions(irr1[2]:-system,irr1[2]:-points);
```

```
(1) [x = -1.01896486399796-1.33092346330283*I, y =
-1.57349962005932+.86187770780964*I, z =
1.0-.906089000592713e-16*I, zz1 =
.801744344776168e-16+.104102228345155e-15*I]

(2) [x = 1.24838627747489-.59841477216379*I, y =
.808428382967238+.924080358318904*I, z =
1.0+.259880445936573e-15*I, zz1 =
.130604196067113e-15+.135595403619785e-15*I]
```

3.5 phc[factor]

A particular case of the problem of numerical decomposition is that of decomposing a hypersurface or, in other words, factoring a multivariate polynomial. *PHCpack* is equipped with a function that treats this special case, see [17] for a description of the algorithms. In *PHCmaple*, it is called `factor`:

```
> f := factor(z*x-z*y);
```

$$\begin{aligned}f := [&0.720227296223279\,10^{-16}\,x + 0.211865299121440\,10^{-15}\,I\,x + z\\&+ 0.727428125630827\,10^{-16} + 0.149806122770695\,10^{-15}\,I, x\\&+ 0.201713788067337\,10^{-15}\,z + 0.184128102532389\,10^{-15}\,I\,z\\&- 1.00000000000000\,y + 0.109119260262696\,10^{-15}\,I\,y\\&+ 0.871486557602219\,10^{-17} - 0.177034993342480\,10^{-15}\,I]\end{aligned}$$

The Maple command `fnormal` is very useful to remove tiny errors in the output:

```
> fnormal(f);
```

$$[z,\ x - 1.000000000\,y]$$

We must point out that the problem of the absolute factorization of polynomials with approximate coefficients has received a lot of research attention, e.g.: [2], [3], [4], [5], [6], [10], since this problem was listed as one of the open challenge problems in [7].

4 Download and Install

PHCmaple has been developed under Windows in the classic worksheet version of Maple 9. In principle, it ought to work with Maple version 8 and higher on most platforms. *PHCmaple* is available for download at `www.math.uic.edu/~leykin/PHCmaple/`. This web page also contains the installation instructions and online help pages for the package.

The purpose of *PHCmaple* is to make the homotopy continuation tools of *PHCpack* available from a general purpose computer algebra system Maple. This package is also the first step of a larger project aimed at creating hybrid symbolic-numeric software for polynomial system solving.

References

[1] E.L. Allgower and K. Georg. *Numerical Continuation Methods, an Introduction*, volume 13 of *Springer Ser. in Comput. Math.* Springer–Verlag, 1990. Reprinted in the SIAM Classics in Applied Mathematics Series, 2004.

[2] R. Corless, M. Giesbrecht, M. van Hoeij, I. Kotsireas, and S. Watt. Towards factoring bivariate approximate polynomials. In B. Mourrain, editor, *Proceedings of ISSAC 2001*, pages 85–92. ACM, 2001.

[3] R. Corless, A. Galligo, I. Kotsireas, and S. Watt. A geometric-numeric algorithm for factoring multivariate polynomials. In T. Mora, editor, *Proceedings of ISSAC 2002*, pages 37–45. ACM, 2002.

[4] A. Galligo and D. Rupprecht. Semi-numerical determination of irreducible branches of a reduced space curve. In B. Mourrain, editor, *Proceedings of ISSAC 2001*, pages 137–142. ACM, 2001.

[5] A. Galligo and D. Rupprecht. Irreducible decomposition of curves. *J. Symbolic Computation*, 33(5):661–677, 2002.

[6] S. Gao, E. Kaltofen, J. May, Z. Yang, and L. Zhi. Approximate factorization of multivariate polynomials via differential equations. In J. Gutierrez, editor, *Proceedings of ISSAC 2004*, to appear.

[7] E. Kaltofen. Challenges of symbolic computation: my favorite open problems. *J. Symbolic Computation*, 29(6):891–919, 2000.

[8] T.Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In F. Cucker, editor, *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, pages 209–304. North-Holland, 2003.

[9] A. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems.* Prentice-Hall, 1987.

[10] T. Sasaki. Approximate multivariate polynomial factorization based on zero-sum relations. In B. Mourrain, editor, *Proceedings of ISSAC 2001*, 284–291. ACM, 2001.

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

[12] A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM J. Numer. Anal.*, 38(6):2022–2046, 2001.

[13] A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical irreducible decomposition using projections from points on the components. In E.L. Green, S. Hoşten, R.C. Laubenbacher, and V. Powers, editors, *Symbolic Computation: Solving Equations in Algebra, Geometry, and Engineering*, volume 286 of *Contemporary Mathematics*, pages 37–51. AMS, 2001.

[14] A.J. Sommese, J. Verschelde, and C.W. Wampler. Using monodromy to decompose solution sets of polynomial systems into irreducible components. In C. Ciliberto, F. Hirzebruch, R. Miranda, and M. Teicher, editors, *Application of Algebraic Geometry to Coding Theory, Physics and Computation*, pages 297–315. Kluwer Academic Publishers, 2001. Proceedings of a NATO Conference, February 25 - March 1, 2001, Eilat, Israel.

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

[16] A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical Irreducible Decomposition using PHCpack. In *Algebra, Geometry and Software Systems*, edited by M. Joswig and N. Takayama, pages 109-130, Springer-Verlag 2003.

[17] A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical factorization of multivariate complex polynomials. *Theoretical Computer Science* 315(2-3): 651-669, 2004. Special Issue on Algebraic and Numerical Algorithms edited by I.Z. Emiris, B. Mourrain, and V.Y. Pan.

[18] H.J. Stetter. *Numerical Polynomial Algebra.* SIAM, 2004.

[19] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software* 25(2):251–276, 1999. Software available at `http://www.math.uic.edu/~jan`.