PHCpack: A Software Library for Polynomial Homotopy Continuation

Jan Verschelde

Department of Math, Stat & CS
University of Illinois at Chicago
Chicago, IL 60607-7045, USA

email: `jan@math.uic.edu`

URL: `http://www.math.uic.edu/~jan`

10 May 2007, Maplesoft

Outline

A. functionality

root counts for sparse and dense systems

platform for numerical algebraic geometry

B. modular design

organization of the code into directories

6 layers built on top of each other

C. interfaces

interactive menus allow scripts in PHCmaple and PHClab

simple `use_c2phc` interface made for use with MPI

A. Functionality of PHCpack

- As blackbox solver: `phc -b input output`, or use `phc -a`.
- Toolbox, for example, as `phc -b` runs in four stages:
  1. `phc -s`: coefficient and equation scaling
  2. `phc -r`: root counting and start system construction
  3. `phc -p`: track paths defined by homotopy
  4. `phc -v`: refine roots and deflate singularities
- On multiprocessor machines: `mpirun -np 8 mpi2track`.

Building phc

- the software is self contained
- compiles with `gcc` (gnu-ada compiler)
- `make phc` works for
  1. Linux (mainly debian)
  2. Windows 2000
  3. Sun Solaris
  4. IBM AIX
  5. MacOS PPC
- alternative compilers: Rational, Aonix, Janus

Homotopy Continuation

Homotopy methods create a family of systems, a so-called homotopy. Typically, to solve $f(\mathbf{x}) = \mathbf{0}$, we construct a start system $g(\mathbf{x}) = \mathbf{0}$ and consider for some random constant $\gamma \in \mathbb{C}$:

$$h(\mathbf{x}, t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}, \text{ for } t \text{ going from 0 to 1.}$$

Continuation methods are used to track the paths $\mathbf{x}(t)$ defined by the homotopy $h(\mathbf{x}(t), t) = 0$. Typically, for a "good" homotopy, singular solutions occur only as $t \approx 1$.

The input coefficients are considered as approximate numbers in $\mathbb{C}$.

By default, all calculations are done with hardware double precision floating-point numbers.

Dense Polynomial Systems

Oversimplifying, looking at n equations in n unknowns defined by $f = (f_1, f_2, \ldots, f_n)$, we assume that

- Dense systems are expected to have as many isolated solutions as predicted by the (multihomogeneous) theorem of Bézout.
- Then the start system $g(\mathbf{x}) = \mathbf{0}$ is

$$\left\{ \begin{array}{cc} x_1^{d_1} - c_1 = 0 & d_1 = \deg(f_1) \\ x_2^{d_2} - c_2 = 0 & d_2 = \deg(f_2) \\ \vdots & \vdots \\ x_n^{d_n} - c_n = 0 & d_n = \deg(f_n), \end{array} \right. \quad \text{with random } c_i \in \mathbb{C}.$$

Or, more generally, every equation in g is a product of linear polynomials. Then we call g a linear-product start system.

Sparse Polynomial Systems

Continuing the oversimplification, still looking at n equations in n unknowns defined by $f = (f_1, f_2, \ldots, f_n)$, we assume that

- Sparse systems have fewer roots than in the dense case.

  Bernshteĭn's theorem states that the mixed volume of the Newton polytopes of f bounds the number of isolated solutions in $(\mathbb{C} \setminus \{0\})^n$.

- Polyhedral homotopies are used to solve a system $g(\mathbf{x}) = \mathbf{0}$ with the same Newton polytopes as f and exactly as many regular solutions as the mixed volume.

For many applications, computing mixed volumes is much easier than tracking that many solution paths. As a general strategy, `phc -b` computes both Bézout bounds and the mixed volume.

Witness Sets

Consider N equations $f = (f_1, f_2, \ldots, f_N)$ in n unknowns.

Suppose $f(\mathbf{x}) = \mathbf{0}$ has k-dimensional solution set V.

Choose k hyperplanes L with random complex coefficients.

Solve $F(\mathbf{x}) = \begin{cases} f(\mathbf{x}) = \mathbf{0} \\ L(\mathbf{x}) = \mathbf{0}. \end{cases}$

observe:

$$F(\mathbf{z}) = \mathbf{0} \quad \Rightarrow \quad \mathbf{z} \in V \cap L$$

moreover:

$$\deg(V) = \#\{\ \mathbf{z} \in \mathbb{C}^n \mid F(\mathbf{z}) = \mathbf{0}\ \}.$$

$\boxed{\text{A witness set } W_L \text{ for } V \text{ consists in } F = (f, L) \text{ and } F^{-1}(\mathbf{0}).}$

Solving Overdetermined Polynomial Systems

- embedding

$$E(f)(\mathbf{x}, z) = \left\{ \begin{array}{r} f_1(x_1, x_2) + \gamma_1 z = 0 \\ f_2(x_1, x_2) + \gamma_2 z = 0 \\ c_0 + c_1 x_1 + c_2 x_2 + z = 0 \end{array} \right. \qquad \begin{array}{l} \text{z is a slack variable} \\ \\ \#\text{slacks} = \text{dimension} \end{array}$$

$\gamma_1, \gamma_2, c_0, c_1, c_2 \in \mathbb{C}$ are random numbers

- cascade

$$h(\mathbf{x}, z, t) = (1-t)E(f)(\mathbf{x}, z) + t \left(\left\{ \begin{array}{r} f_1(x_1, x_2) + \gamma_1 z = 0 \\ f_2(x_1, x_2) + \gamma_2 z = 0 \\ z = 0 \end{array} \right. \right).$$

The cascade starts at the top dimension. Solutions with $z \neq 0$ are regular and are the start solutions for $h(\mathbf{x}, z, t) = \mathbf{0}$.

Filtering and Factoring

- membership test

  Given point $\mathbf{p} \in \mathbb{C}^n$ and witness set W_L for V, does $\mathbf{p} \in V$?

$$h(\mathbf{x}, t) = (1-t) \left(\begin{array}{c} f(\mathbf{x}) \\ L(\mathbf{x}) \end{array} \right) + t \left(\begin{array}{c} f(\mathbf{x}) \\ L(\mathbf{x}) - L(\mathbf{p}) \end{array} \right) = \mathbf{0}.$$

  Test: $\mathbf{p} \in V \cap L'$? where $L' = L - L(\mathbf{p})$.

- monodromy loops

  Consider two witness sets W_L and W_K for V and the homotopy

$$h(\mathbf{x}, \gamma, t) = \gamma(1-t) \left(\begin{array}{c} f(\mathbf{x}) \\ L(\mathbf{x}) \end{array} \right) + t \left(\begin{array}{c} f(\mathbf{x}) \\ K(\mathbf{x}) \end{array} \right) = \mathbf{0}.$$

  Choose $\gamma = \alpha$ for t going from 0 to 1 and let $\gamma = \beta$ for t going from 1 to 0. Points on same irreducible component permute.

diagonal homotopies and `phc -a`

In $\mathbb{C}^3$, given two witness sets W_A and W_B for two 2-dimensional sets A and B, cut out respectively by by L_{A1}, L_{A2} and L_{B1}, L_{B2}. Consider $h(\mathbf{x}, t) =$

$$(1-t)\left(\begin{array}{c} f_A(u_1,u_2,u_3) = 0 \\ f_B(v_1,v_2,v_3) = 0 \\ L_{A1}(u_1,u_2,u_3) = 0 \\ L_{A2}(u_1,u_2,u_3) = 0 \\ L_{B1}(v_1,v_2,v_3) = 0 \\ L_{B2}(v_1,v_2,v_3) = 0 \end{array}\right) + t \left(\begin{array}{c} f_A(u_1,u_2,u_3) = 0 \\ f_B(v_1,v_2,v_3) = 0 \\ u_1 - v_1 = 0 \\ u_2 - v_2 = 0 \\ u_3 - v_3 = 0 \\ L_{AB}(u_1,u_2,u_3) = 0 \end{array}\right).$$

As t goes from 0 to 1, the deformation starts at pairs $(\alpha, \beta) \in A \times B$, where α and β are witness point on A and B. At $t = 1$ we find witness points on the curve of intersection.

```
Welcome to PHC (Polynomial Homotopy Continuation) V2.3.26  1 May 2007

Running in full mode.  Note also the following options:
  phc -0 : random numbers with zero seed for repeatable runs
  phc -a : Solving polynomial systems equation-by-equation
  phc -b : Batch or black-box processing
  phc -c : Irreducible decomposition for solution components
  phc -d : Linear and nonlinear Reduction w.r.t. the total degree
  phc -e : SAGBI/Pieri homotopies to intersect linear subspaces
  phc -f : Factor pure dimensional solution set into irreducibles
  phc -k : realization of dynamic output feedback placing poles
  phc -l : Witness Set for Hypersurface cutting with Random Line
  phc -m : Mixed-Volume Computation via lift+prune and MixedVol
  phc -p : Polynomial Continuation by a homotopy in one parameter
  phc -q : Tracking Solution Paths with incremental read/write
  phc -r : Root counting and Construction of start systems
  phc -s : Equation and variable Scaling on system and solutions
  phc -v : Validation, refinement and purification of solutions
  phc -w : Witness Set Intersection using Diagonal Homotopies
  phc -z : strip phc output solution lists into Maple format
Options may be combined, e.g.: phc -b -0 or phc -0 -b.
```

B. Programming Style

- PHCpack is written in Ada
- chosen C for recent main programs, calling Ada code:
  1. routines for pole placement, with Yusong Wang
  2. parallel factorization, with Anton Leykin
  3. parallel polyhedral homotopies, with Yan Zhuang
- key tool is "package" to implement

  libraries: as in classic mathematical software libraries;

  classes: object oriented programming.

  A package has a specification (interface) and a body (implementation) in two separate files.

Version 1.0 of PHCpack was developed in Ada 83 (archived by ACM TOMS), rewritten using Ada 95, released as version 2.

Modular Structure of PHCpack

- The experimental computational laboratory aspect of PHCpack implies that several alternative methods coexist.
  Often useful for independent verification and testing.
- The code is organized in a hierarchy of directories.
  Every directory (or "module") targets a specific function:
  e.g.: path following, mixed volumes, etc.
- Each module contains
  1. packages to implement classes or libraries;
  2. interactive programs to test the packages;
  3. driver routines called by the main program.

Testing Strategies

The testing of the code happens at three levels:

in the small: relative small test programs allow the user to given specific input or generate random inputs

in each module: drivers organize the functions offered by the packages in a module, accessible via menus

at the large: do `phc -b` on a large collection of "demo" polynomial systems with known output (benchmarking)

Six Layers in PHCpack

1. Basic Data Structures and Operations
2. Homotopy, Newton, and Continuation
3. Root Counts and Start Systems
4. Numerical Schubert Calculus
5. Tools for a numerical irreducible decomposition
6. Interfaces, Main program, and use of MPI

Modular Design: the directories

6	CtoPHC, Main, Lib, MPI	6
	Components:	
5	Samplers, Interpolators, Factorization, Decomposition, Solver	5
4	SAGBI, Pieri, Induction	4
	Root_Counts:	
3	Product, Fewnomials, Implift, Stalift, Dynlift, Symmetry, MixedVol	3
2	Homotopy, Newton, Continuation	2
	Math_Lib:	
1	System; Numbers, Matrices, Polynomials, Supports	1

1. Basic Data Structures and Operations

The System module contains facilities to time programs. Different makefiles pick different timers, depending on Windows or not.

The mathematical library makes PHCpack self contained.

features: linear algebra and efficient polynomial evaluation over general number fields, mainly in floating point arithmetic.

in progress: more robust parsing of the input from file and from strings, also Laurent polynomials (negative exponents) accepted on input

plans: integrate ATLAS, LAPACK, and faster multiprecision arithmetic; apply methods from algorithmic differentiation

2. Homotopy, Newton, and Continuation

The secondary main data structure is a polynomial system with a corresponding list of solutions to define a start system. Newton's method is the basic validation tool, recently augmented with deflation, and it is used as corrector in path following methods.

features: a collection of path following methods independent of the kind of homotopy

in progress: 1. integrate deflation into the main solvers

2. deal with quadratic turning points in a real sweep

plans: apply Smale's α-theory to compute certificates

3. Root Counts and Start Systems

A root count bounds the number of expected isolated solution. Based on the root count, a start system that has exactly as many regular solutions as the root counted is constructed and solved.

features: linear-product start systems for dense and polyhedral methods for sparse systems, recently started incorporation of MixedVol (ACM TOMS Algorithm 845), developed by Tangan Gao, Tien-Yien Li, Xing Li, and Mengnien Wu

in progress: efficient evaluation of linear-product systems; improved numerical stability of the polyhedral homotopies

plans: exploit structure when dealing with solution sets

4. Numerical Schubert Calculus

Enumerative geometry is a classical branch of algebraic geometry. Pieri gave a geometric proof of Bézout's theorem.

features: Pieri homotopies solve the output pole placement problem of linear systems control

in progress: implementation of Littlewood-Richardson homotopies based on Ravi Vakil's recent geometric proof

plans: efficient numerical tools to deal with determinantal equations

5. Tools for a numerical irreducible decomposition

Numerical algebraic geometry aims to relate to algebraic geometry as numerical linear algebra to linear algebra.

features: witness sets computed via homotopy cascade; monodromy certified by linear traces factors pure dimensional solution sets into irreducible components; diagonal homotopies to intersect witness sets; an equation-by-equation solver

in progress:
1. parallel implementation of subsystem-by-subsystem solver
2. apply deflation for singular solution sets

plans: find exceptional sets of solutions for specific parameters

6. Interfaces, Main program, and use of MPI

PHCpack leads to a menu driven and file oriented program `phc`. Using MPI has led to a programmer's interface to PHCpack.

features: interactive menus and tools used via scripts in PHCmaple and PHClab

in progress: bring structure to the C library of wrappers around `use_c2phc`; Python interface

plans: enable wider and flexible use of PHCpack; better and more adequate documentation

C. Interfaces to PHCpack

1. scripts walk through menus of phc

   thanks to Nobuki Takayama (OpenXM) for this idea

2. C program prepares input, then calls Ada program

   after computations, call C program to process results

3. but a third interface is needed to implement parallel programs using MPI in a good way

Applying Program Inversion to Homotopy Solver

$$h(\mathbf{x}, t) = \gamma(1-t)g(\mathbf{x}) + t f(\mathbf{x}) = \mathbf{0}, \quad \gamma \in \mathbb{C}, \quad t \in [0,1].$$

Input: $g(\mathbf{x}) = \mathbf{0}$;
for k from 1 to $\#g^{-1}(\mathbf{0})$ do
    compute $\mathbf{y}_k$: $g(\mathbf{y}_k) = \mathbf{0}$;
end for;
output: $g^{-1}(\mathbf{0})$.

solve start system

track paths to target

Input: $g^{-1}(\mathbf{0})$, $h(\mathbf{x}, t) = \mathbf{0}$;
for k from 1 to $\#g^{-1}(\mathbf{0})$ do
    path starts at $\mathbf{y}_k \in g^{-1}(\mathbf{0})$;
end for;
output: $f^{-1}(\mathbf{0})$.

Applying Program Inversion to Homotopy Solver

$$h(\mathbf{x}, t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}, \quad \gamma \in \mathbb{C}, \quad t \in [0,1].$$

Input: $h(\mathbf{x}) = \mathbf{0}$;
for k from 1 to $\#g^{-1}(\mathbf{0})$ do
  path starts at $\mathbf{y}_k \in g^{-1}(\mathbf{0})$;
end for;
output: $f^{-1}(\mathbf{0})$.

track paths to target

get next start solution

Input: $g(\mathbf{x}) = \mathbf{0}$, k;
compute $\mathbf{y}_k$: $g(\mathbf{y}_k) = \mathbf{0}$;
  or read $\mathbf{y}_k$ from file;
  or type in values for $\mathbf{y}_k$;
output: $\mathbf{y}_k \in g^{-1}(\mathbf{0})$.

one gateway routine

```
extern void adainit( void );
extern int _ada_use_c2phc ( int task, int *a, int *b, double *c );
extern void adafinal( void );
```

main parallel programs deliberately written in C, using MPI

Input for PHCpack

The main structured input data for PHCpack are polynomials.

Three ways to enter polynomials:

1. reading polynomials from file;
2. polynomials are parsed from strings;
3. given term after term and add up.

Goal: avoid that the user program must duplicate the effort of building data structures for multivariate polynomials.

PHCpack as State Machine

- consider a vending machine:
  1. make selection
  2. push button
  3. collect product
- data stored in “containers”
  1. polynomials read from file or added term after term;
  2. solutions written to file or enumerated.
- job numbers define meaning of the parameters
- encapsulation of low level `use_c2phc` via library

a subroutine in phc_solve

```
int input_output_on_files ( void )
{
   int fail,rc;

   fail = syscon_read_system();
   printf("\nThe system in the container : \n");
   fail = syscon_write_system();
   fail = solve_system(&rc);
   printf("\nThe root count : %d\n",rc);
   printf("\nThe solutions :\n");
   fail = solcon_write_solutions();

   return fail;
}
```

another subroutine

```
int interactive_input_output ( void )
{
   int n,fail,k,nc,i,rc;    char ch,p[80];
   printf("\nGive the number of polynomials : "); scanf("%d",&n);
   fail = syscon_initialize_number(n);
   printf("\nReading %d polynomials, ",n);
   printf("terminate each with ; (semicolon)...\n");
   for(k=1; k<=n; k++)
   {
      printf("-> polynomial %d : ",k); ch = getchar();
      read_poly(&nc,p); fail = syscon_store_polynomial(nc,n,k,p);
   }
   fail = solve_system(&rc);
   printf("\nThe root count : %d\n",rc);
   printf("\nThe solutions :\n"); fail = solcon_write_solutions();
   return fail;
}
```