phcpy: an API for PHCpack

Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
http://www.math.uic.edu/~jan
jan@math.uic.edu

Graduate Computational Algebraic Geometry Seminar

Outline

1. Introduction
   - an Application Programmers Interface for Python
   - design of a programmers interface to PHCpack
2. Functionality
   - regression testing on examples
   - the core: phcpy
   - containers for persistent sessions
   - Pieri homotopies, algebraic sets, and client/server interactions
3. Implementation
   - building on the API for the C programmer
   - code for the Python extension module
   - documentation generated with Sphinx

phcpy: an API for PHCpack

1. Introduction
   - an Application Programmers Interface for Python
   - design of a programmers interface to PHCpack
2. Functionality
   - regression testing on examples
   - the core: phcpy
   - containers for persistent sessions
   - Pieri homotopies, algebraic sets, and client/server interactions
3. Implementation
   - building on the API for the C programmer
   - code for the Python extension module
   - documentation generated with Sphinx

terminology and definitions

Python is a very popular scripting language:

- well suited to teach programming and computer science,
- in widespread use with a strong developers community,
- provides a mechanism to add modules and packages.

PHCpack provides algorithms for solving polynomial systems:

- PHC = Polynomial Homotopy Continuation,
- blackbox solver `phc -b` gives one solver,
- many other uses available through menus.

An API is an Application Programmers Interface:

- a Python programmer can build a specific solver,
- good for benchmarking and regression testing.

motivations

Some good reasons for `phcpy`:

1. Sage is a free and open source alternative to the big M's.
   Sage contains `phc.py` in the source directory
   `devel/sage-main/sage/interfaces`.

2. With `numpy`, `scipy`, `sympy`, and `matplotlib`
   we do scientific and symbolic computing in Python,
   `ipython --pylab` is an enhanced Interactive Python.

3. Reproducibility: re-compute results of published papers.
   In computational science, a paper is an advertisement.
   Can *others* reproduce *your* computational results?

4. The web interface to PHCpack. Instead of launching `phc`
   through input and output files, we better import `phcpy`.

development history

- Pre-history: use of Message Passing Interface (MPI) to call the path trackers of PHCpack leads to the `use_c2phc` gateway. In joint work with Yusong Wang, Yan Zhuang, Yun Guan, and Anton Leykin, the main program was always a C program.

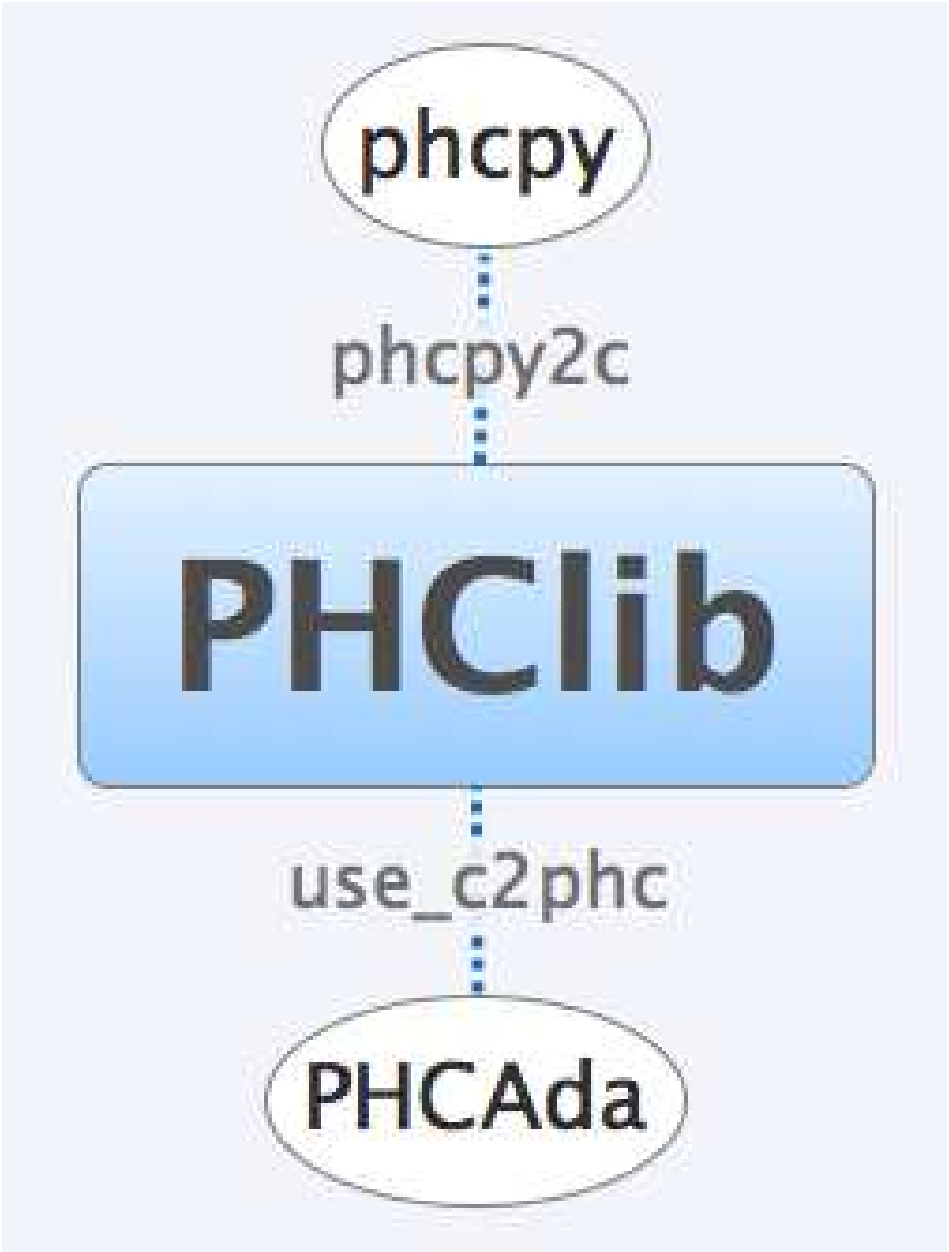
- In Fall 2006, at the IMA workshop on *software for algebraic geometry*, Kathy Piret and William Stein make a Python interface. The current version of `phc.py` in Sage dates from 2008 is written by Marshall Hampton and Alex Jokela.

  The PhD thesis of Kathy Piret of 2008 describes the module `phcpy.py`. This module exports the blackbox solver, a mixed volume calculator, and a path tracker.

- Lecture 40 of MCS 507 (30 November 2012) introduces Sphinx (a documentation generation tool) using `phcpy` as an example.

CyPHC: Marc Culler uses Cython to interface to PHCpack.

a simple mind map



PHCpack as a state machine

phcpy is build on top of the C interface to PHCpack.

The C interface was developed for the parallel versions of the path trackers in PHCpack.

- The main program, written in C, does the scheduling of the path tracking jobs, using the Message Passing Interface (MPI).
- Every job makes calls to the path trackers of PHCpack, the code written in Ada.

The C interface should remain light, without having to duplicate the data structures for polynomials and solutions.

Idea: we can build up a polynomial adding one term at a time, with calls to an interface routine.

persistent sessions with PHCpack

As a consequence of using PHCpack as a state machine,
we can hold persistent sessions.

For example, calling the solver proceeds in three stages:

1. parse the input strings into polynomials,
2. apply the solver to the stored polynomials,
3. extract list of solutions as strings.

The three stages are three calls to PHCpack.
Each stage alters the internal state.

The data passed by the function calls are *persistent*.
The data are not lost after the execution of the function ends.

the current state of phcpy

Currently `phcpy` has version 0.0.5 and date 2012-12-30.

There are six modules:

- `examples.py` : some interesting benchmark examples
- `phcpy.py` : use of C extension module phcpy2c
- `phcsols.py` : PHCpack solution string into dictionary
- `schubert.py` : numerical Schubert calculus
- `phcsets.py` : manipulate algebraic sets
- `phcwulf.py` : client/server for beowulf clusters

phcpy: an API for PHCpack

1. Introduction
   - an Application Programmers Interface for Python
   - design of a programmers interface to PHCpack

2. Functionality
   - regression testing on examples
   - the core: phcpy
   - containers for persistent sessions
   - Pieri homotopies, algebraic sets, and client/server interactions

3. Implementation
   - building on the API for the C programmer
   - code for the Python extension module
   - documentation generated with Sphinx

the module `examples`

The module `examples` contains ten benchmarking examples.
The examples are stored as strings in functions of the module.

Use in two different ways:

1. regression test: `python examples.py`

   Each system `s` is solved with `s = phcpy.solve(f)` and tested with `assert len(s) == n` where `n` is the known #solutions.

2. interactive use: import test examples

```
>>> from examples import noon3
>>> f = noon3()
>>> for p in f: print p
...
x1*x2^2 + x1*x3^2 - 1.1*x1 + 1;
x2*x1^2 + x2*x3^2 - 1.1*x2 + 1;
x3*x1^2 + x3*x2^2 - 1.1*x3 + 1;
```

running the blackbox solver

```
>>> from examples import noon3
>>> f = noon3()
>>> from phcpy import solve
>>> s = solve(f)

ROOT COUNTS :

total degree : 27
3-homogeneous Bezout number : 29
  with partition : {x1 }{x2 }{x3 }
general linear-product Bezout number : 21
  based on the set structure :
     { x1 }{ x2 x3 }{ x2 x3 }
     { x1 x3 }{ x1 x3 }{ x2 }
     { x1 x2 }{ x1 x2 }{ x3 }
mixed volume : 21
stable mixed volume : 21
```

the script `solve` in `phcpy.py`

```
def solve(L,silent=False):
    """
    Calls the blackbox solver of PHCpack.
    On input in L is a list of strings.
    By default, the solver will print to screen the
    computed root counts.  To make the solver silent,
    set the flag silent to True.
    """
    py2c_syscon_clear_Laurent_system()
    py2c_solcon_clear_solutions()
    n = len(L)
    m = py2c_syscon_initialize_number_of_Laurentials(n)
    for i in range(0,n):
        p = L[i]
        nc = len(p)
        py2c_syscon_store_Laurential(nc,n,i+1,p)
    rc = py2c_solve_Laurent_system(silent)
    return load_solutions()
```

containers for persistent sessions

The two main data structures handled by PHCpack are polynomial systems and lists of solutions.

There is a systems and a solutions container.

The container approach:

1. initialize with string representations,
2. function calls use container data,
3. results are printed from container data.

Information hiding: specific data structures are hidden.

other core functions of phcpy

- Mixed volumes and polynomial homotopies (Bernshteĭn):
  - `mixed_volume()` is an interface to ACM TOMS Algorithm 846 of Tangan Gao, T.Y. Li, Mengnien Wu, and Li Xing.
    The option `stable` counts solutions with zero coordinates.
  - `random_coefficient_system()` runs the polyhedral homotopies to solve a system with random coefficients.
- Linear-product root counts and start systems (Bézout):
  - `linear_product_root_count()` generates a linear-product structure using the degrees of the given system.
  - `random_linear_product_system()` return a start system that has as many solutions as the linear-product Bézout count.
- Running `track()` on target, start systems and start solutions returns all end points of the paths (artificial-parameter homotopy).
- `newton_step()` does one step with Newton's method to produce solution diagnostics such as condition number and residual.

running tests

Typing `python phcpy.py` at the command prompts runs

1. `test_polyhedral_homotopy()`
   A polyhedral homotopy solves a random coefficient system.
   This function allows to monitor the progress of the solver using the container of all mixed cells in a regular subdivision constructed to compute the mixed volume.

2. `test_solver()`
   Two random "trinomials" are generated and solved.
   A trinomial has (at most) three monomials in every equation.
   If the constant term is absent, then the stable mixed volume is strictly larger than the mixed volume.

3. `test_track()`
   Tests the tracking of paths between two random trinomial systems.

the module `phcsols.py`

The command `s = phcpy.solve(f)` returns in `s` a list of strings.

The loop

```
>>> for sol in s: print sol
```

shows the list of solutions in the PHCpack format.

The module `phcsols`

- converts the PHCpack solution strings into dictionaries,
- evaluates the solutions by plain substitution.

Running `python phcsols.py` demonstrates its usage.

numerical Schubert calculus

`schubert.py` exports Pieri homotopies to solve the problem:
given (m, p, q),

- m is the dimension of the input planes in $\mathbb{C}^{m+p}$,
- p is the dimension of the output planes in $\mathbb{C}^{m+p}$,
- q is the degree of the maps that produce output planes,

$mp + q(m+p)$ input m-planes and $mp + q(m+p)$ interpolation points,
compute all maps of degree q that produce p-planes nontrivially intersecting the given m-planes at the interpolation points.

Three parts in `schubert.py`:

1. a root count gives the number of solutions in the generic case,
2. Pieri homotopies solve a generic instance,
3. cheater homotopies solve a fully real specific problem.

`python schubert.py` solves a fully real instance of the problem.

algebraic sets

The module `phcsets.py` exports tools to manipulate positive dimensional solution sets of polynomial systems.

Currently, `phcsets.py` offers

- A function to embed a polynomial system.
  An embedding adds extra hyperplanes and slack variables.
- A basic version of a factorization method applying monodromy.

`python phcsets.py` applies monodromy factorization to a curve.

client/server interaction

The module `phcwulf.py` offers a simple client/server interaction to solve many random trinomial systems.

The client/server interactions works as follows:

- The server maintains a list of polynomial systems and allows for n clients to make requests.
- Client i solves system at position k, where $i = k \mod n$. This is a static work load balancing.

The computational server in the web interface to PHCpack needs a more advanced client/server interaction.

phcpy: an API for PHCpack

1. Introduction
   - an Application Programmers Interface for Python
   - design of a programmers interface to PHCpack
2. Functionality
   - regression testing on examples
   - the core: phcpy
   - containers for persistent sessions
   - Pieri homotopies, algebraic sets, and client/server interactions
3. Implementation
   - building on the API for the C programmer
   - code for the Python extension module
   - documentation generated with Sphinx

the use_c2phc function

The Ada function:

```
function use_c2phc ( job : integer;
                     a : C_intarrs.Pointer;
                     b : C_intarrs.Pointer;
                     c : C_dblarrs.Pointer ) return integer;
```

The corresponding C function:

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

One uniform streamlined design of the interface:

- The C programmer calls one single Ada function `use_c2phc`.
- What `use_c2phc` executes depends on the job number.
- The `(a,b,c)` parameters are flexible enough to pass strings.

the structure of PHClib

The API for C programmers was developed for parallel programming, using MPI (Message Passing Interface).

PHClib is a collection of C wrappers around `use_c2phc`.

Layers in PHClib:

1. Collections of C functions (documented by `.h` files) wrap the `use_c2phc` as a C library.
2. Small programs in C test a specific functionality. For example, `phc_solve` calls the blackbox solver.
3. The main parallel programs with MPI call the C wrappers.

PHClib contains `phcpy2c.h` and `phcpy2c.c`, needed to build the extension module and the shared object `phcpy2c.so`.

the shared object phcpy2c.so

The first line of code in phcpy is `from phcpy2c import *`
where phcpy2c is defined by the shared object phcpy2c.so.

The file phcpy2c.h contains the line

```
static PyObject *py2c_solve_system
 ( PyObject *self, PyObject *args );
```

defined by the code in phcpy2c.c:

```
static PyObject *py2c_solve_system
 ( PyObject *self, PyObject *args )
{
   int fail,rc;

   initialize();
   if(!PyArg_ParseTuple(args,"")) return NULL;
   fail = solve_system(&rc);
   return Py_BuildValue("i",rc);
}
```

documentation generated with Sphinx

What is Sphinx? From `http://sphinx-doc.org`:

- Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license.
- It was originally created for the new Python documentation, and it has excellent facilities for the documentation of Python projects, but C/C++ is already supported as well, and it is planned to add special support for other languages as well.
- Sphinx uses reStructuredText as its markup language, and many of its strengths come from the power and straightforwardness of reStructuredText and its parsing and translating suite, the Docutils.

The *same* documentation is generated in several formats, including latex, pdf, html.

Why Sphinx? Used for the python language, numpy, sympy, matplotlib.

still so many things to do...

The current version of phcpy is only at 0.0.5.

Increased functionality is needed, some examples:

- callback functions to the path trackers,
- better factorization and diagonal homotopies,
- working with monomial maps and tropical methods, ...

Technical issues:

- group modules in one package,
- installation bundle with easy install egg.