The Gift Wrapping Method in 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. Gift Wrapping
   - a geometric algorithm to compute convex hulls
   - outline of the algorithm and data structures
2. Implementation in PHCpack
   - overview of the code
   - the module `polytopes` of `phcpy`

Outline

1. Gift Wrapping
   - a geometric algorithm to compute convex hulls
   - outline of the algorithm and data structures

2. Implementation in PHCpack
   - overview of the code
   - the module `polytopes` of `phcpy`

the convex hull problem

A polytope can be defined in two ways:

1. the convex hull of finitely many points (*V-representation*), or
2. the intersection of finitely many half planes (*H-representation*).

Special cases:

1. the points span a space of lower than the expected dimension,
2. the intersection of half planes is unbounded.

The convex hull problem:

given the V-representation, compute the H-representation.

By duality, consider normals to facets as points.

gift wrapping to compute the convex hull

One of the very first algorithms to compute convex hulls:

- D.R. Chand and S.S. Kapur.: **An algorithm for convex polytopes.** *Journal of the Association for Computing Machinery*, 17(1):78–86, 1970.
- G. Swart. **Finding the convex hull facet by facet.** *Journal of Algorithms*, 6:17–48, 1985.
- K.H. Borgwardt. **Average complexity of a gift-wrapping algorithm for determining the convex hull of randomly given points.** *Discrete Comput. Geom.*, 17(1):79–109, 1997.

For simplicial polytopes, average complexity $\sim$ linear programming.

Dynamic programming may speedup the calculations.

the f-vector of a polytope

The f-vector of a d-dimensional polytope P:

$$(f_0(P), f_1(P), \ldots, f_{d-1}(P)), \quad f_k(P) = \#k\text{-dimensional faces of } P.$$

The Euler-Poincaré formula: $\displaystyle\sum_{k=-1}^{d} (-1)^k f_k(P) = 0$, $f_{-1}(P) = f_d(P) = 1$.

Let $M_d(t) = (t, t^2, \ldots, t^d)$, *a cyclic d-polytope with n vertices*

$$C(n,d) = \mathrm{conv}\left(\{M_d(t_1), M_d(t_2), \ldots, M_d(t_n)\}\right),$$

for n distinct choices of $t_1, t_2, \ldots, t_n$.

For $2k \leq d$: $f_k(C(n,d)) = \displaystyle\binom{n}{k+1}$.

For any k: $f_k(C(n,d)) = O\left(\lfloor d/2 \rfloor ! n^{\lfloor d/2 \rfloor}\right) \geq f_k(P)$, for any polytope P.

specifications for the implementation

For Newton polytopes of sparse polynomials:

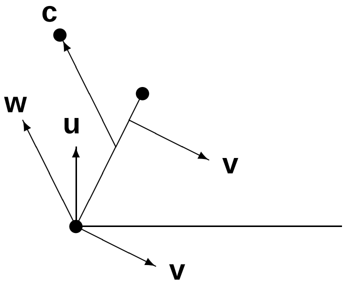
1. relatively few points,
2. exact integer arithmetic is preferred.

Input/output specifications:

- The algorithm is recursive, with the base case a polygon.
  Input: $A \in \mathbb{Z}^{2 \times n}$. Output: $V \in \mathbb{Z}^{2 \times m}$.
  The columns of V are the coordinates of the m vertices of $\mathrm{conv}(A)$, oriented counterclockwise, consecutive columns span edges.
- For $A \in \mathbb{Z}^{3 \times n}$, the output is list of facets. Data for each facet:
  1. the facet normal $\mathbf{v}$;
  2. the vertex points that span the facet $\mathrm{conv}(\mathrm{in}_{\mathbf{v}} A)$; and
  3. connecting pointers: every edge of $\mathrm{conv}(\mathrm{in}_{\mathbf{v}} A)$ has a pointer to the unique neighboring facet.

the main property for gift wrapping

Every $(d-2)$-dimensional face is the intersection of two facets.



Assume $\mathbf{u} = (0,0,1)$ is the inner normal of $\mathrm{in}_{\mathbf{u}} P$.
Take an edge $\mathrm{in}_{\mathbf{v}} A$ with an inner normal **v** perpendicular to **u**.
All points of A lie above the plane spanned by $\mathrm{in}_{\mathbf{u}} A$.
The point **c** which spans jointly with $\mathrm{in}_{\mathbf{v}} A$ the neighboring facet to $\mathrm{in}_{\mathbf{u}} A$ lies at the end of a **w** that makes the largest possible angle with **v**.

Outline

1. Gift Wrapping
   - a geometric algorithm to compute convex hulls
   - outline of the algorithm and data structures

2. Implementation in PHCpack
   - overview of the code
   - the module `polytopes` of `phcpy`

outline of the algorithm

A $(d-2)$-dimensional face of d-dimensional polytope is *a ridge*.
For $d = 3$, a ridge is an edge.

A graph traversal algorithm proceeds in three steps:

1. Compute an initial facet: the root node.
2. Compute the ridges of a node.
3. Given a node and a ridge, compute the other node that connects to the given node at the given ridge.

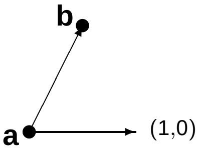
computing an initial facet

Consider a supporting hyperplane to the polytope

1. touching first at the lexicographically lowest point **a**,
2. and then supporting the initial edge.

Think of the supporting hyperplane as a piece of wrapping paper.

The initial edge is spanned by **a** and the point **b** for which the angle of the vector $\mathbf{a} - \mathbf{b}$ with $(1, 0, \ldots, 0)$ is largest.



3. For a facet in 3-space, we need to compute a third point **c**:
    1. **c** is not collinear with **a** and **b**; and
    2. $\langle \mathbf{c}, \mathbf{v} \rangle = m$, where $\mathbf{v} \perp \mathbf{a} - \mathbf{b}$ and $m = \min\limits_{\mathbf{a} \in A} \langle \mathbf{a}, \mathbf{v} \rangle$.

storing the graph structure

For a three dimensional polytope, we store a list of facets.

For each facet, we store

1. a unique label as the identification number of the facet,
2. the inner normal to the facet, with components of the vector normalized so their greatest common divisor equals one,
3. labels to the vertex points that span the facet,
4. for each edge of the facet, a pointer to the neighboring facet.

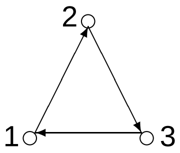
A facet in 3-space spanned by n vertices has n edges.

For facets in 4-space, we store the ridges of each facet along with pointers to the neighbors instead of the 4-th item in the list above.

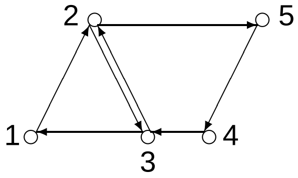
connecting adjacent facets

Vertices of facets are ordered counterclockwise.

For example, the facet spanned by $(1, 2, 3)$:



The facet spanned by $(1, \overrightarrow{2, 3})$ is adjacent
to the facet spanned by $(\overrightarrow{3, 2}, 5, 4)$.



computing an adjacent facet

Given a facet, to geometrically compute an adjacent facet:

- take the supporting hyperplane passing through the facet,
- choose one edge (or a ridge in dimension > 3) of the facet,
- rotate the hyperplane around the edge (or ridge) till it meets the next vertex.

Consider the supporting hyperplane as a piece of wrapping paper. As we wrap the polytope, we fold the paper over an edge.

To compute the adjacent facet that shares a particular edge, we must find the vertex that makes the widest angle between

1. the inner normal to the facet, and
2. a vector perpendicular to the edge ending at that vertex.

Outline

overview of the code

The gift wrapping method for Newton polytopes is available

- with operations in 64-bit aritmetic,
- in arbitrary multiprecision integer arithmetic.

Currently available are methods to compute the convex hull of Newton polytopes in the plane, in 3-space and 4-space.

The module `polytopes` of `phcpy` exports a function to compute the convex hull of points in the plane.

a numerical example

Ten points randomly generated with values in $\{-9, \ldots, +9\}$:

```
 -1  6  7  8 -9  4  9  6  8  6
  3 -3 -2 -6 -8  9 -8  2  3  4
 -6  2  2 -4 -6  6 -6  6 -3 -6
```

There are 13 facets, 20 edges, and 9 vertices.

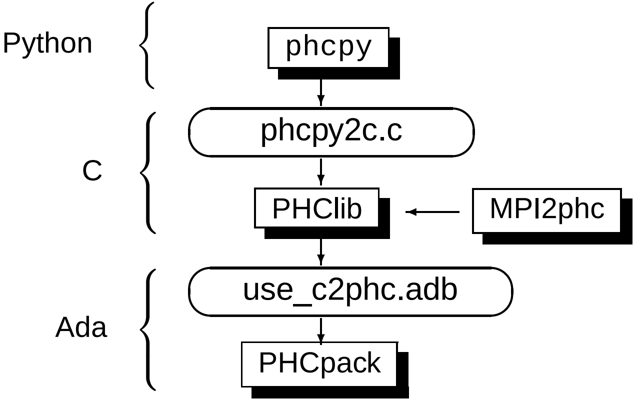
The initial facet:

```
facet 0 spanned by 5 1 6 has normal 132 -96 -7 \
  and value -378
  IP : -378 1066 1102 1660 -378 -378 1998 558 789 450 \
  support : 1 5 6
  neighboring facets : 1 2 3
```

The first edge:

```
edge 0 is intersected by 0 0 1 and 132 -96 -7
  vertex 5 belongs  vertex 1 belongs
```

the design of phcpy



Outline

1. Gift Wrapping
   - a geometric algorithm to compute convex hulls
   - outline of the algorithm and data structures

2. Implementation in PHCpack
   - overview of the code
   - the module `polytopes` of `phcpy`

a random example

```
$ python
>>> import phcpy
>>> from phcpy.polytopes import random_points
>>> pts = random_points(2,5,-9,9)
>>> pts
[(-7, 3), (5, -6), (-5, 7), (-4, 6), (6, 1)]
>>> from phcpy.polytopes import planar_convex_hull
>>> (v, n) = planar_convex_hull(pts)
>>> v
[(6, 1), (-5, 7), (-7, 3), (5, -6)]
>>> n
[(-6, -11), (2, -1), (3, 4), (-7, 1)]
```

ongoing and future work

Wrapping of code in PHCpack to extend `phcpy`:

- Construct the list of facets in 3-space and 4-space.
- Query the data structures in two ways:
  1. enumerate with `get_next_{vertex, edge, ridge, facet}`,
  2. walk on the polytope, to adjacent facets, ridges, edges, vertices.

Complete the implementation:

- Convex hulls for point configurations in any dimension.
- Let `get_next_facet()` launch the computation, giving the user of `phcpy` control over the order of execution.
- Extend the gift wrapping method to compute pretropisms.