

Standardization in computational topology

Jānis Lazovskis, University of Latvia



Open problem session
ICERM, May 20, 2026

Standardization in computational topology

Personal context: An issue I have struggled with for the past couple years, especially this year

General context: You find / create a mathematical method, and you want to make a computer do it (for exploration, publication, collaboration, etc). Then...

one of the following happens (in no particular order):

- ▶ You implement it yourself without issues
- ▶ You get someone else to do it, they implement it
- ▶ You attempt to implement it yourself, it takes longer than expected
- ▶ You get someone to help you implement it, it takes longer than expected
- ▶ You are unable to implement it exactly, you find an existing, not-quite-the-same method
- ▶ You are unable to implement it exactly, you give up

Reasons: Obscure code, incompatible packages, unclear / misleading documentation, ...

Standardization in computational topology

Technical reasons for not succeeding: Code you work with often requires specific data types, assumes some PH steps have been done, or executes the PH pipeline in its own particular way:

- ▶ reading in / transforming the data
- ▶ reducing the matrix
- ▶ getting persistence pairs / representative cycles

This “common stuff” makes it harder for you to implement your “interesting stuff”

Problem: Every PH software does the basic (PH) operations from scratch, in its own way.
Users unfamiliar with the software waste time* in adjusting, redoing, verifying.
Barriers grow to getting new people involved.

Would be great if: You could focus on implementing your specific contribution, without making annoying wrapper / helper functions, knowing the precise data types of what you need

cat-list.github.io

The Computational and Applied Topology List

Search

Hint: Press enter to add the current search term as a tag filter.

Chosen tags:

All tags:

complex/alpha + complex/cech + complex/cell + complex/chromatic-alpha + complex/cluster-tree + complex/cover-mapper + complex/cubical + complex/cw + complex/delay-coordinate + complex/digraph + complex/dwaker + complex/flsa +

↓ Show more ↓

Software	Tags
2pac	complex/rips + custom/boundary-matrix + custom/distance + custom/filtration + lang/c++ + lang/cli + lang/python + type/multiparameter + type/persistence +
bats	complex/cell + complex/cubical + complex/rips + complex/simplicial + complex/witness + custom/distance + custom/filtration + lang/c++ + lang/python + type/persistence + type/relative-homology + type/representative + type/vineyard + type/zigzag +
cereberus	complex/simplicial + lang/python + type/Reeb + type/mapper + type/mergetree +
chomp	complex/cubical + complex/simplicial + custom/boundary-matrix + lang/c++ + type/discrete-morse-theory + type/persistence +

The current situation

https://numpy.org/doc/stable/reference/arrays.interface.html

The array interface protocol

Note

This page describes the NumPy-specific API for accessing the contents of a NumPy array from other C extensions. [PEP 3118](#) - [The Revised Buffer Protocol](#) introduces similar, standardized API for any extension module to use. [Cython](#)'s buffer array support uses the [PEP 3118](#) API; see the [Cython NumPy tutorial](#).

version: 3

The array interface (sometimes called array protocol) was created in 2005 as a means for array-like Python objects to reuse each other's data buffers intelligently whenever possible. The homogeneous N-dimensional array interface is a default mechanism for objects to share N-dimensional array memory and information. The interface consists of a Python-side and a C-side using two attributes. Objects wishing to be considered an N-dimensional array in application code should support at least one of these attributes. Objects wishing to support an N-dimensional array in application code should look for at least one of these attributes and use the information provided appropriately.

This interface describes homogeneous arrays in the sense that each item of the array has the same "type". This type can be very simple or it can be a quite arbitrary and complicated C-like structure.

```
typedef struct {
    int two; /* contains the integer 2 -- simple sanity check */
    int nd; /* number of dimensions */
    char typekind; /* kind in array -- character code of typestr */
    int itemsize; /* size of each element */
    int flags; /* flags indicating how the data should be interpreted */
    /* must set ARR_HAS_DESCR bit to validate descr */
    Py_ssize_t *shape; /* A length-nd array of shape information */
    Py_ssize_t *strides; /* A length-nd array of stride information */
    void *data; /* A pointer to the first element of the array */
    PyObject *descr; /* NULL or data-description (same as descr key
                    of __array_interface__) -- must set ARR_HAS_DESCR
                    flag or this will be ignored. */
} PyArrayInterface;
```

An analogous solution

Defining a protocol

Proposition: Our community should have a **collection of rules** that describes the types of “common data” used in computational topology, and specifies their properties.

- ▶ *Similar to NumPy's dense matrix interface protocol, but for sparse matrices*
- ▶ *Similar* to OAT's oracles for user queries, but at the level of computer queries*

Simple (personally suggested) example:

- ▶ A *matrix* is a *dictionary* with $\mathbf{Z}_{\geq 0}$ indices and non-empty $\mathbf{Z}_{\geq 0}$ -*vectors*.
 - ▶ A *barcode* is an *unordered set* whose elements are *pairs* of elements of $\mathbf{Z}_{\geq 0}$.
-

Things to keep in mind:

- ▶ People who implement major packages should have an open discussion about the different technical issues. *Who are these people? How can they be brought together?*
- ▶ Lots of software is already developed, people are happy with their own setup. *How can people be convinced this is worthwhile?*