Intro. to Gudhi

Tao Hou

Sketch

- Overview
- Installation
- Python API

Gudhi overview

- From official website (https://gudhi.inria.fr/):
  - *a generic open source **C++ library**, with a **Python interface**,*
  - *for Topological Data Analysis (**TDA**) and Higher Dimensional Geometry Understanding.*
  - *offers state-of-the-art data structures and algorithms to construct simplicial complexes and compute persistent homology*
- Developed by Inria from France

C++ vs. Python

- C++ programs: Fast to execute, hard to develop
- Python programs: Slow in execution, easy to develop
- Gudhi was developed by C++, i.e., all the intensive computations are in C++; you can also use Gudhi directly from C++
- Provides Python interface
  - You can use the Gudhi C++ codes from Python
  - *Fast* to execute, *easy* to develop
  - Pybind11, Cython

Installation

- The official website recommended using Conda
  - https://gudhi.inria.fr/python/latest/installation.html
  - You can use both GUI and commandline to install packages on Conda
- For installing Conda on linux (especially ubuntu, debian):
  - If sudo apt-get doesn't work (e.g., if you don't have access to the writing permission of the root directory)
  - Use wget to get the installation script and bash run the script:
    - wget https://repo.anaconda.com/archive/Anaconda3-2023.07-2-Linux-x86_64.sh
    - bash Anaconda3-2023.07-2-Linux-x86_64.sh
    - The link can be retrieved from: https://www.anaconda.com/download/

Gudhi python interface

Documentation

- https://gudhi.inria.fr/python/latest/

Categories of the API

- Cell complex data structures
  - **Simplicial complex**, **Cubical complex**
- Filtrations
  - Alpha complex, **Rips complex**, Witness complex, Cover complex (Mapper, Nerve and Graph Induced Complexes)
- **Compute persistent homology**
- Persistent barcode processing
  - Bottleneck distance, Wasserstein distance, Persistence representations (vectorizations and kernels for ML)
- **Persistence graphical tools**
  - display persistence barcode, diagram or density
- Others: Point cloud utilities (read from file, subsample, find neighbors), generating datasets, etc.

Simplicial complex (in math)

- A collection of sets, where each set contains vertices
- A set *a* is a subset of a set *b*: *a* is a **face** of *b* and b is a **coface** of *a*

SimplexTree

- https://gudhi.inria.fr/python/latest/simplex_tree_user.html
- https://gudhi.inria.fr/python/latest/simplex_tree_ref.html
- Jean-Daniel Boissonnat and Clément Maria. The simplex tree: an efficient data structure for general simplicial complexes. Algorithmica, pages 1–22, 2014.

SimplexTree

Example

```
import gudhi
st = gudhi.SimplexTree()
if st.insert([0, 1]):
    print("[0, 1] inserted")
if st.insert([0, 1, 2], filtration=4.0):
    print("[0, 1, 2] inserted")
if st.find([0, 1]):
    print("[0, 1] found")
result_str = 'num_vertices=' + repr(st.num_vertices())
print(result_str)
result_str = 'num_simplices=' + repr(st.num_simplices())
print(result_str)
print("skeleton(2) =")
for sk_value in st.get_skeleton(2):
    print(sk_value)
```

- https://gudhi.inria.fr/python/latest/simplex_tree_user.html

Building a SimplexTree

- Build from scratch using insert():
  - Declare a SimplexTree object
  - Insert a simplex to the complex: automatically add the faces of the simplices
- You can assign a filtration value when you insert a simplex:
  - All complexes in Gudhi can be *filtered*, e.g., an object of SimplexTree can always represent a filtration for the complex

$$\mathcal{F} : \varnothing = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$$

$$K_i \text{ and } K_{i-1} \text{ differ by one simplex denoted by } \sigma_i^{\mathcal{F}}$$

- More often, we would not build a SimplxTree (or a filtration) from scratch like above, but would rather build a filtration (i.e., an SimplexTree object) from a filtration building class (such as RipsComplex)

Simplices in a SimplexTree

- A simplex is represented as a python list of integers:
  - A tetrahedron (3-simplex): [5, 9, 10, 17]
  - Every simplex returned from SimplexTree, and every simplex feed into SimplexTree as parameter is a list of integers
  - For a bunch of simplices, then it is a list of lists of integers
- Advantage: Claritity, no confusion
- This also makes mapping a simplex to a certain property a little tricky (e.g., you build a dual graph for a surface, and want to look up the vertex id corresponding to a triangle)
  - Would have to use a python dictionary (e.g., {‘A’: 1, ‘B’: 2})
  - The key for the dictionary is the simplex (the list of ints)
  - Have to convert the list into a tuple (`tuple(l)`) or a string (`str(l)`) first so that `l` can be used as a key for the dict

Simplices in a SimplexTree

- Some more info: in the C++ interface, each simplex has an integer id; you can manipulate a simplex simply by its id
- You can get its list of vertices by invoking a method
- This can be convenient in certain cases
- The python interface can be a little tricky if you switch from it C++ version at first

Adjacency info. in SimplexTree

- get_simplices()
  - Get all the simplices
- get_boundaries()
  - Get the boundary simplices for a simplex
- get_cofaces()
  - Get all the cofaces of a simplex (the star)
  - Get the cofaces in a certain dimension for a simplex

RipsComplex (class for building a Rips filtration in Gudhi)

- Rips filtration:
  - Given a set of points (point cloud) with pair-wise distance for the points, it returns you a filtration
  - The persistence of the filtration gives you an idea of the topology of the underlying space of the point cloud
- You can create a RipsComplex in two ways in Gudhi:
  - By specifying coordinates for the points
  - By providing a distance matrix
- What you eventually get from the class is a **SimplexTree object** representing the rips filtration (using the create_simplex_tree() method)
- https://gudhi.inria.fr/python/latest/rips_complex_user.html

Compute persistence barcode for a SimplexTree

- persistence():
  - Return type: list of pairs (dimension, pair(birth, death))
  - (Don’t have time to verify, but) the returned pair is a pair of filtration value (i.e., a **persistence interval**), e.g., for Rips filtration, it is the distance where a homology class is born and the distance where the homology class dies (not the index of the simplices in the simplex-wise filtration)
- persistence_pairs():
  - another form of persistence barcode, where each pair is a pair of simplices that generates a persistence interval
  - however, we don’t really the indices of the simplices in the simplex-wise filtration (this can be useful but I don’t find a way to do this in the current python interface of Gudhi)
  - note: you can definitely do this in the C++ interface

Compute persistence barcode for a SimplexTree

- You can also compute the **extended persistence** by extended_persistence(), which is equivalent to the **level zigzag persistence** (in a sense)
- Cohen-Steiner, David, Herbert Edelsbrunner, and John Harer. "Extending persistence using Poincaré and Lefschetz duality." *Foundations of Computational Mathematics* 9.1 (2009): 79-103
- Carlsson, Gunnar, Vin De Silva, and Dmitriy Morozov. "Zigzag persistent homology and real-valued functions." *Proceedings of the twenty-fifth annual symposium on Computational geometry*. 2009

Displaying persistence barcode

- https://gudhi.inria.fr/python/latest/persistence_graphical_tools_user.html
- If you want to explore some more advanced displaying options, you would have to know more about `matplotlib`

Useful but shall not go into details

- Persistent barcode processing
  - Bottleneck distance
  - Wasserstein distance
  - Persistence representations (vectorizations and kernels for ML)
- Some other features provided by Gudhi python (see the user manual)

CubicalComplex

- A cell complex consisting of vertices (dim-0), edges (dim-1), squares (dim-2), cubes (dim-3)
- Very useful for processing images (2-complex) and 3D volume datasets (3-complex)
- A filtration is built from a **function** from a regular 2D/3D grid to the real values
- https://gudhi.inria.fr/python/latest/cubical_complex_user.html
- https://gudhi.inria.fr/python/latest/cubical_complex_ref.html

A very important (and tricky) point about CubicalComplex

- Two ways for constructing it:
  - Either specify an array named '**top_dimensional_cells'** – A multidimensional array of top dimensional cells filtration values
  - or specify an array named '**vertices'** – A multidimensional array of vertices filtration values
- The first corresponds to a 'toplex filtration': It's a filtration where you add the top-dimensional cells, or toplexes (e.g., cubes in 3D) one by one, and the lower dimensional cells will be added as needed
- The second correspond to the lower star filtration for a PL function: where you give function values to the vertices, and values on other points are interpolated
- The persistence of the two ways of interpreting a cubical dataset is not too different indeed

Compute persistence for CubicalComplex

- Similar to SimplexTree, use persistence()

About the C++ interface of Gudhi

- Much more powerful
- Learning curve is steeper
- Besides the reference manual, you should also thoroughly study the examples provided

Some other libraries I know (or have used)

- Ripser (https://github.com/Ripser/ripser): good for computing rips filtrations
- Dionysus (https://mrzv.org/software/dionysus2/): also very versatile (e.g., vineyard); provide both C++ and Python interfaces
- Phat (https://github.com/blazs/phat): very fast in computing persistence; tricky to build
- Homcloud (https://homcloud.dev/index.en.html): Good for computing representatives for persistence
- FZZ (https://github.com/taohou01/fzz): fast for zigzag persistence; only in C++; uses phat
- More: https://cat-list.github.io/