CS233: The Shape of Data Homework 2
Geometric and Topological Data Analysis
Stanford University Wednesday, 22 April 2026

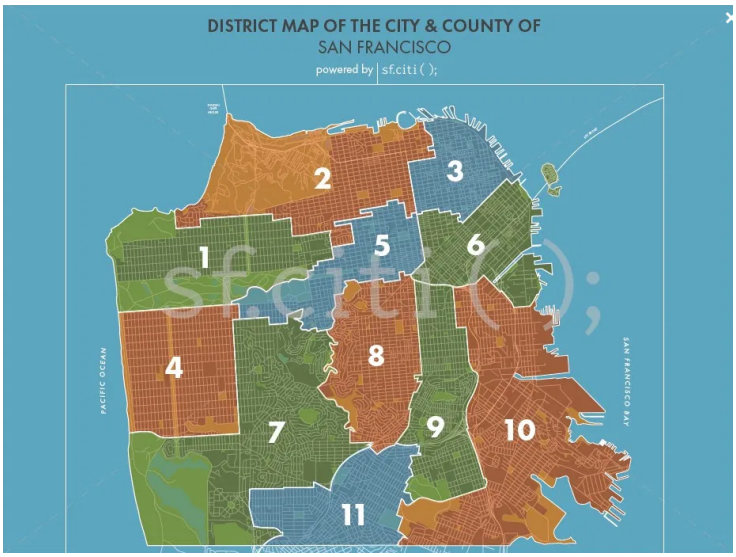
Homework #2: Computational Topology - Simplicial Complexes, Homology, Persistence, Bottleneck Distances and the Mapper Algorithm [90 points total. *Note that Problem 4 comes in two versions, 4a and 4b, with the same number of points – choose only one.*]
Due Date: Wednesday, May 6, 2026

Problem 1. [10 points]

A simplicial complex is a combinatorial structure used to encode topological information on point-cloud data, typically based on proximity information. A simplicial complex can also be used to encode the topology of various spatial partitions.

Consider the decomposition of San Francisco into 11 districts as shown below (from `http://www.sfcity.org`). Create an abstract simplicial complex to represent the 11 regions and their topological adjacencies. Each region will become a vertex (0-simplex) of the complex, pairs of regions that share a common boundary edge give rise to edges (1-simplices), and triplets of regions that have a common point become triangles (2-simplices of the complex).

1. Draw a schematic diagram of your simplicial complex embedded in the plane. You may orient the simplices in any reasonable way, e.g., edges go from west to east, and all triple region intersections go around counterclockwise. Indicate the orientations with arrows. (The east-west relationship and the adjacency of districts can sometimes be ambiguous. Any reasonable interpretation will work.)



2. Write down the boundary matrices corresponding to your complex: ∂_1 is a matrix with rows the 0-simplices and columns the 1-simplices; ∂_2 is a matrix with rows the 1-simplices and columns the 2-simplices. Verify that $\partial_1\partial_2 = 0$ (with $\mathbb{Z}_2 = \mathbb{Z}/2\mathbb{Z}$ arithmetic), as claimed.

3. A *filtration* of a simplicial complex K is a sequence of subcomplexes such that i) $\emptyset = K^0 \subset K^1 \subset \cdots \subset K^m = K$, ii) $K^{i+1} = K^i \cup \sigma^{i+1}$ where σ^{i+1} is a simplex of K.

   Extend the numbering of the regions (i.e., the complex vertices) given in the image to a full *filtration* of the complex by assigning to each simplex the largest value of its vertices and ordering simplices of the same value by dimension. Write down the sequence $\sigma^1, \sigma^2, \ldots, \sigma^m$ of simplices in the way they would be added one by one in the filtration.

Problem 2. [20 points]

Environment Setup We will use Python and some topology libraries for Problems 2-4. We recommend creating a dedicated conda environment by running the following (also provided in `install.sh`)

```
conda create -n cs233hw2 python=3.11
conda activate cs233hw2
pip install ripser trimesh gudhi==3.8.0 networkx
```

Extraction of topological information from point cloud data is the main focus of this problem; we will explore this in two scenarios:

- Points sampled from a shape in Euclidean space.
- After adding a film of background noise to sampled points.

In this problem, we will use **Ripser.py**[1], a persistent homology package for Python. To get familiar, start by going through the first section of the demonstration[2] on computing persistence diagrams with Risper. Use this knowledge to work through and report results for the following exercises:

1. Generate a sampling of a unit circle in the plane (100 sample points) and verify that the persistence diagram has a single non-noise point in the 1-dimensional homology.

2. Generate a pair of circles and draw 100 sample points from each circle. Experiment with overlapping and non-overlapping circles. Can you recover the expected topology? Try adding a small Gaussian noise to the sampled points and observe the results.

[1] `https://ripser.scikit-tda.org/`

[2] `https://ripser.scikit-tda.org/en/latest/notebooks/Basic%20Usage.html#`

3. Generate a noisy circle lying in a square film of background noise (e.g. 100 circle points + 100 background points). Discuss how easy or difficult it is to bring out the correct topological signal in the diagram with the added background noise.

Pick an integer $1 \leq k \ll n$. The $k-$codensity $\text{cod}_k(x)$ of a point x is the distance from x to its k^{th} nearest neighbor. If $\text{cod}_k(x)$ is small, then x is a crowded region of the data. This is implemented for you in function `codensity`, which takes as input the $n \times n$ matrix `dist` containing pairwise distances between all points.

4. How successfully can you use this codensity to extract the topology of the circle lying in background noise from the previous part?

We now extend the idea of measuring topology on a circle to a sphere of a general dimension k. The goal is to calculate the homology of the $k-$sphere in all dimensions:

5. Generate N points uniformly sampled on the surface of a $k-$sphere in $\mathbf{R}^{k+1}$ (by generating points in a spherically symmetric normal distribution and then normalizing the corresponding vectors to unit length). To what extent can you robustly recover the Betti profile $[1,0,\ldots,0,1,0,\ldots]$? How large can we make k so that reproducible results can no longer be obtained? Plot the persistence diagrams of different (k,n) you've tried.

   (You may try N around 100 and increase k until it takes ≥ 20 minutes.)

6. Apply a background film of uniform Gaussian noise, with mean -2 and standard deviation 4 as follows:

```
noise = -2 + 4 * np.random.randn(M, k + 1)
noised_data = np.concatenate([data, noise])
```

   Here, `data` is the set of points sampled on the k-sphere in the previous problem. Using a codensity estimator to filter out the noise, can you still recover the correct Betti sequence for the sphere? Try with $k > 1$. Plot the computed persistence diagram before and after denoising.

Problem 3. [30 points]

Shape features used to perform shape analysis tasks can be extrinsic or intrinsic. Extrinsic features are features that depend on the embedding of the shape in 3D, while intrinsic features are dependent only on measurements made on the surface of the object. To see the distinction, consider the articulations of humans or mammals. These are near-isometric deformations in the sense that they preserve geodesic distances on the shape surface, which are intrinsic features. By contrast, extrinsic features, such as Euclidean distances or shape poses can drastically change under such deformations.

Using geodesic distances, we can view shapes as just metric spaces. The Gromov-Hausdorff distance is a metric that measures how far two compact metric spaces are from being isometric.

For near-isometric shapes we expect it to be very close to zero. We first define the Hausdorff distance, and then use this notion to define the Gromov-Hausdorff distance.

Hausdorff distance: Let X and Y be two non-empty subsets of a metric space (M,d). We define their Hausdorff distance $d_H(X,Y)$ to be

$$d_H^M(X,Y) = \max\{\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y)\} \tag{1}$$

Gromov-Hausdorff distance: The Gromov-Hausdorff distance between compact metric spaces (X,d_X) and (Y,d_Y) is:

$$d_{GH}((X,d_X),(Y,d_Y)) = \inf_{Z,\gamma_X,\gamma_Y} d_H^Z(\gamma_X(X),\gamma_Y(Y)), \tag{2}$$

where γ_X, γ_Y range over all isometric embeddings of X, Y into some shared metric space (Z,d_Z).

An alternate definition of the Gromov-Hausdorff distance is given as follows:
For sets A and B, a subset $R \subset A \times B$ is a *correspondence* (between A and B) if and only if
(1) $\forall a \in A$, *there exists* $b \in B$, *s.t.* $(a,b) \in R$
(2) $\forall b \in B$, *there exists* $a \in A$, *s.t* $(a,b) \in R$
Let $\mathscr{R}(A,B)$ denote the set of all possible correspondences between sets A and B.

Consider metric spaces (X,d_X) and (Y,d_Y). Let $\Gamma : X \times Y \times X \times Y \to \mathbf{R}^+$ be given by $\Gamma(x,y,x',y') = |d_X(x,x') - d_Y(y,y')|$. Then, the Gromov-Hausdorff distance between X and Y can be rewritten as

$$d_{GH}(X,Y) := \frac{1}{2}\inf_{R\in\mathscr{R}(A,B)}\max_{(x,y)\in R,(x',y')\in R}\Gamma(x,y,x',y'). \tag{3}$$

The above definition provides a view of the Gromov-Hausdorff distance as obtaining the specific correspondence that minimizes the maximum discrepancy of distance differences between corresponding points in the two shapes. To read more about the Gromov-Hausdorff distance metric and for a proof showing the equivalence of the two definitions, you can check the book[3] by D. Burago, Y. Burago and S. Ivanov.

In the following problem, we use an Euclidean space as (Z,d_Z). Even in this setting, computing Gromov-Hausdorff distances can be computationally demanding. We begin with a couple of toy examples to become familiar with these distances:

1. Compute the Hausdorff distance between two circles in the Euclidean plane of radii r_1, r_2 and centered at $(0,0)$ and $(d,0)$ respectively. Compute an upper bound to the Gromov-Hausdorff distance of the two circles by computing the Hausdorff distance between them after a translation that makes both these circles concentric. What do you observe as $r_1 \to r_2$?

[3] `https://www-ams-org.stanford.idm.oclc.org/books/gsm/033/gsm033-endmatter.pdf`

You have learned about Rips complexes of finite metric spaces in class, as well as about persistence diagrams and their bottleneck distances. An interesting fact is that the bottleneck distance between the persistence diagrams of the Rips filtrations of two finite metric spaces can be proven to be a lower bound on their Gromov-Hausdorff distance. Thus the bottleneck distance can become a "poor man's" substitute for the expensive GH distance. This is shown in the paper "Gromov Hausdorff Stable Signatures for Shapes using Persistence" by Chazal, Cohen-Steiner, Guibas, Memoli and Oudot[4].

Following the convention of the paper, here we use a slightly different definition of the Rips complex $R^\alpha(L)$, where α now represents the radius instead of the diameter of the ball. Formally, the Rips complex $R^\alpha(L)$: for $p_0,\ldots,p_k \in L$, $\sigma = [p_0 p_1 \cdots p_k] \in R^\alpha(L) \iff \forall i,j, \in \{0,\cdots,k\}, d(p_i,p_j) \leq \mathbf{2}\alpha$.

2. Compute (by hand) the bottleneck distance between 0-dimensional persistence diagrams of the Rips complex on the vertices of a k-sided regular polygon inscribed in a circle of radius r_1, and the Rips complex on the vertices of a k-sided regular polygon inscribed within a circle of radius r_2. How does this bottleneck distance vary in the limit as $k \to \infty$? Is the lower bound assertion satisfied?

3. Consider X, a set of two points at distance 2, and Y, a set of two points at distance $2+2\epsilon$ in the plane. What is the Gromov-Hausdorff distance between the two sets? Sketch the $0-$dimensional persistence diagrams of the two point sets, and compute the bottleneck distance between them. Is the lower bound to the Gromov-Hausdorff distance tight?

For the remainder of this problem, you can use Ripser to obtain the persistence diagram and another Python package **gudhi** to compute the bottleneck distances. Please refer to the user manual[5] to familiarize yourself with bottleneck distance computation with gudhi. Shape data in the form of point clouds will be provided to you in `shapes.npz`. This data consists of both the sampled point collections of 6 shapes, and the inter-nodal Dijkstra distances for the sampled points in the shapes with full resolution (**note:** you can perform all the computations for the following problem without explicitly using the point clouds, and using just the Dijkstra matrix if you use the `distance_matrix` feature in Ripser).

4. Consider the point clouds in `shapes.npz`. These point clouds are representative of two different sets of isometric shapes. Use the persistence algorithm to compute the lower bound on the Gromov-Hausdorff distances between these shape spaces. Depending on the values of the bottleneck distances between these shape spaces, classify these point clouds into two different classes of isometric shapes.

Problem 4. [30 points - Do only 4a or 4b]

Variant 4a — *for those of you more experimentally inclined:*

[4] `http://geometry.stanford.edu/papers/ccgmo-ghsssp-09/ccgmo-ghsssp-09.pdf`
[5] `https://gudhi.inria.fr/python/latest/bottleneck_distance_user.html`

The Mapper algorithm is a tool for topological analysis of geometric data such as point clouds. The algorithm filters the data using a user-provided "lens" function and generates a graph skeleton structure that helps visualize the data, based on both point proximity and the function values. Please refer to lecture slides for a detailed description of the algorithm.

In this problem, you are provided with a partial implementation of the Mapper algorithm. You will complete the partial implementation, and then apply the algorithm to a set of shapes to perform a shape analysis task.

Mapper implementation: The input to the algorithm is a shape mesh S with vertex set V. The "lens" or feature function $f : V \to [a,b] \subset \mathbf{R}$ maps mesh vertices to real values. We use the z coordinate of vertices as an example. The algorithm has a couple of parameters: the number of intervals n_{ints} and the interval size s. The start points of the intervals are $a, a+h, a+2h, \ldots, a+(n-1)h$, where $h = \frac{b-a}{n_{ints}}$, and the end points are $a+s, a+h+s, ..., a+(n-1)h+s$ respectively. You can change the variables `ints` and `isize` in the Python implementation to try different values of n_{ints} and s. The defaults are set to `ints = 10` and `isize = 2(b-a)/ints - 1e-6`. Feel free to use your own interval sizes or intervals of unequal sizes, as long as neighboring intervals have nonempty overlap. You may also experiment with any feature function over the shape as the lens function.

The output of your algorithm is a Mapper graph. Each node of the graph represents a connected component on the mesh, obtained as follows. Let G be the Mapper graph, we iterate through all intervals i and incrementally add new nodes representing connected components to G. For i^{th} interval $[a+ih, a+ih+s]$, **compute** $S_i = f^{-1}([a+ih, a+ih+s])$, where f^{-1} is the pre-image of the feature function f. **Compute** the connected components of S_i, denoted by $\Omega_{i,1}, \Omega_{i,2}, ..., \Omega_{i,n_i}$. Add n_i new nodes to G, all labeled i, each representing a connected component $\Omega_{i,j}$. **Add an edge** between the node corresponding to $\Omega_{i-1,k}$ $(k = 1, 2, ..., n_{i-1})$ and node corresponding to $\Omega_{i,j}$, if $\Omega_{i-1,k} \cap \Omega_{i,j}$ is non-empty.

You are provided with 12 human shape meshes, where the z-coordinate corresponds to the normal upright direction. Please complete the following exercises:

1. Complete the missing parts of the provided Python implementation of the Mapper algorithm corresponding to the **bold** words in the description above.

2. Plot the graph for all shapes in folder `shapes`, with the z-coordinate as the lens function. Note that the label should be 1 for shape vertices with the lowest lens function value and n_{ints} for shape vertices with the highest ones. Please attach the plot for shape `victoria0` to your solution.

3. In which of these shapes do the Mapper graphs contain a cycle? Why?

4. Do you observe a pattern in the graphs unique to the shapes with a standing pose? Why? Shapes `victoria2` and `victoria24` are both shapes in the reclining pose. Plot their Mapper graphs with the z coordinate as the feature function. Why do they have a different number of nodes with label 1?

5. **Bonus question:** Is it possible to infer the pose of the shapes from the Mapper graphs alone? Select one or more feature functions that you believe can help in this task. Play around with all parameters to perform this task — interval sizes, interval overlaps, feature functions, etc.

Variant 4b — *for those of you more theoretically inclined:*

Prove that the homology groups H_k for the p-simplex are trivial in dimensions $1 \leq k \leq p$. In other words, show that for any k, $1 \leq k < p$, *any* cycle of k-simplices in the p simplex is in fact a boundary of some chain of $k+1$-simplices (so $Z_k = B_k$). For simplicity you can assume that homology is taken over $\mathbb{Z}_2$ (so each simplex is present with a coefficient of 0 or 1 only).

[*Starting hint:* If all the k-simplices in cycle c do not contain a particular vertex v, then add v to all these k-simplices to form a chain of $k+1$-simplices, whose boundary is the original cycle c.]