Extended Abstract: A Description of the JavaPlex Software Package

Author[1,*] and Author[2]

[1]Affiliation
[2]Affiliation

January 21, 2011

Abstract

In this document, we describe the design and goals of the JavaPlex software package. Its main purpose is to support research in the area of topological data analysis; the two main capabilities being the construction of filtered chain complexes of vector spaces, and the computation of their persistent homology.

1 Motivation and Design Goals

The main reason for the existence of JavaPlex is to provide researchers in the area of topological data analysis a unified software library to support their investigations. With this in mind, the design goals for it are as follows:

- **Support for new directions for research** The main goal of the JavaPlex package is to provide an extensible base to support new avenues for research in computational homology and data analysis. While its predecesor jPlex was very well suited towards computing simplicial homology, its design made extension difficult.
- **Interoperability** JavaPlex can be run either as a Java application, or it can be called from Matlab in jar form. Future possibilities include providing scripting interfaces in bsh or jython.
- **Adherence to generally accepted software engineering practices** As a means to realizing the first goal, the JavaPlex software package was designed and implemented with software engineering best-practices. Emphasis was placed on maintainability, modularity and reusability of the different parts of the code.

We refer the reader to [Car09] for a very readable introduction to the field of topological data analysis as well as the computational tasks involved.

2 Previous Work

The JavaPlex package is the fourth version in the Plex family. These programs have been developed over the past decade by members of the computational topology research group at Stanford University. Each successive version incorporated the results of new advances in the relatively quickly developing fields of computational topology and topological data analysis.

Like JavaPlex, its predecessor jPlex was also written in the Java language. However, it differed in that the main goal of jPlex was the computation of *simplicial* homology. Recent research topics in topological data analysis have required practitioners to move beyond conventional simplicial homology to more general scenarios.

*Contents of footnote

3 Algebraic Background on Persistence Modules

- Maybe the best person to write this would be Mikael or Gunnar??
- need to mention grading by filtration index vs grading by dimension

4 Filtered Complex Generation

As mentioned in the abstract, the first main function of JavaPlex is the construction of filtered chain complexes of vector spaces associated to actual point cloud datasets. The motivation for such constructions is that they provide a persistent model of the dataset in question across all scales. JavaPlex currently supports the consruction of two main types of filtered simplicial complexes: the Vietoris-Rips and lazy-witness constructions. To begin, suppose that we have a finite metric space $(\mathcal{X}, d)$. In practice, it is possible that $\mathcal{X}$ is a set of points in Euclidean space, although this is not necessary.

4.1 The Vietoris-Rips Construction

We define the filtered complex $VR(\mathcal{X}, r)$ as follows. Suppose that the points of $\mathcal{X}$ are $\{x_1, ...x_N\}$, where $N = |\mathcal{X}|$. The Vietoris-Rips complex is constructed as follows:

- **Add points:** For all points $x \in \mathcal{X}$, $x \in VR_0(\mathcal{X}, 0)$
- **Add 1-skeleton:** The 1-simplex $[x_i, x_j]$ is in $VR_1(\mathcal{X}, r)$ iff $d(x_i, x_j) \leq r$
- **Expansion:** We define $VR(\mathcal{X}, r)$ to be the maximal simplicial complex containing $VR_1(\mathcal{X}, r)$. That is, a simplex $[x_0, ..x_k]$ is in $VR(\mathcal{X}, r)$ if and only if all of its edges are in $VR_1(\mathcal{X}, r)$.

An extensive discussion on algorithms for computing the Vietoris-Rips complex can be found in [Zom10]. The JavaPlex implementation is based on the results of this paper.

4.2 The Lazy-Witness Construction

The fundamental idea behind the lazy-witness construction is that a relatively small subset of a point cloud can accurately describe the shape of the dataset. This construction has the advantage of being more resistant to noise than the Vietoris-Rips construction. An extensive discussion about it can be found in [dSC04].

The lazy-witness construction starts with a selection of landmark points, $\mathcal{L} \subset \mathcal{X}$ with $|\mathcal{L}| = L$. One possibility is to simply choose a random subset of $\mathcal{X}$. Another possibility is to perform a sequential max-min selection: An initial point l_0 is selected, and then we inductively select the point l_k which maximizes the minimum distance to all previously generated points. This max-min construction tends to produce more evenly spaced points than the random selection. Again we refer the reader to [dSC04] for a more detailed discussion, as well as empirical results supporting these claims.

This construction is parameterized by a value ν, which most commonly takes the values 0, 1, or 2. We also define the distance matrix D to contain the pairwise distances between the points in $\mathcal{X}$.

- **Define m_i:** If $\nu = 0$, let $m_i = 0$, otherwise, define m_i to be the ν-th smallest entry in the i-th column of D
- **Add points:** For all points $l \in \mathcal{L}$, $l \in LW_0(\mathcal{X}, 0, \nu)$
- **Add 1-skeleton:** The 1-simplex $[l_i, l_j]$ is in $LW_1(\mathcal{X}, r, \nu)$ iff there exists an $x \in \mathcal{X}$ such that $\max(d(l_i, x), d(l_j, x)) \leq r + m_i$.
- **Expansion:** We define $LW(\mathcal{X}, r, \nu)$ to be the maximal simplicial complex containing $LW_1(\mathcal{X}, r, \nu)$.

5 Homology Computation

At the core of the JavaPlex library is the set of algorithms that actually compute the homology of a filtered chain complex. Key references to background material regarding these algorithms can be found in [ZC05, dSMVJ10]. Although we do not describe them in detail here, we note that the algorithms for computing persistent absolute/relative (co)homology can be formulated as matrix decomposition problems. The fundamental reason for this is the equivalence of category of persistent vector spaces of finite type, and the category finitely generated graded modules over $\mathbb{F}[t]$. This correspondence is described in [ZC05].

The homology algorithms are built in a way that is optimized for chain complexes implemented as *streams*. By this we mean that a filtered chain complex is represented by a sequence of basis elements that are produced in increasing order of their filtration indices. Enforcing the constraint that all complexes must be implemented this way allows JavaPlex to perform the matrix decomposition operations in an efficient online fashion.

6 Applications

Although in principle JavaPlex can compute the persistent homology of arbitrary chain complexes of vector spaces, almost always these complexes arise from some sort of topological construction. Below we outline these different situations.

6.1 Simplicial Homology

This is the "standard" situation, which was also handled by previous versions in the Plex family. Here, we have a filtered sequence of simplicial complexes $X_1 \subset X_2 \subset ... \subset X_n$, from which we define the vector space of chains, $C(X_i)$ consisting of formal sums of elements of X_i with coefficients in the field $\mathbb{F}$.

In this case the boundary operator $\partial : C(X_i) \to C(X_{i-1})$ is the actual geometric boundary defined by

$$\partial([v_0, ..., v_n]) = \sum_i (-1)^i [v_0, ..., \hat{v_i}, ..., v_n]$$

6.2 Cellular Homology

This is where we depart from previous Plex implementations. In this case, $X = X_*$ is a filtered cell complex. This complex is formed by inductively adding n-cells to the $n-1$-skeleton, by the gluing maps

$$\varphi_\alpha^n : S^{n-1} \to X_{d-1}$$

which map the boundaries of the n-cells e_α^n to the $n-1$ skeleton. Note that in the above, we use n to denote the grading by dimension, and d to denote the grading by filtration index.

The boundary operator then becomes

$$\partial(e_\alpha^n) = \sum_\beta \deg(\varphi_{\alpha\beta}^n) e_\beta^{n-1}$$

where deg refers to the topological degree of a map.

Note that while JavaPlex is fully capable of computing the persistent homology of arbitrary cell complex, the specification of such complexes are more tedious than simplicial complexes. Nevertheless they offer the user a parsimonious way of defining a wide class of topological spaces.

6.3 Operations on Chain Complexes

The abstraction away from geometric primitives allows JavaPlex to handle more general algebraic constructions using complexes.

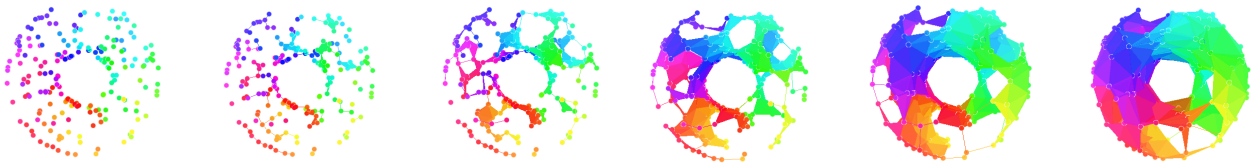

Figure 1: Example of a Lazy-Witness complex generated from randomly sampled points on a torus.

- **Tensor Products:** Given two chain complexes (graded by dimension), (A_*, d_*) and (B_*, d'_*), the tensor product complex $(A \otimes B)_*$ is defined by

$$(A \otimes B)_n = \bigoplus_{p+q=n} A_p \otimes B_q$$

  Given two such complexes, the construction of the tensor product is very straightforwardly implemented in JavaPlex.

- **Hom Complex:** Given two chain complexes (graded by dimension), (A_*, d_*) and (B_*, d'_*), the hom-complex $\mathrm{Hom}(A, B)_*$ is defined by

$$\mathrm{Hom}(A, B)_n = \bigoplus_{p \in \mathbb{Z}} \mathrm{Hom}(A_p, B_{p+n})$$

  which in the case of field coefficients reduces to

$$\mathrm{Hom}(A, B)_n = \bigoplus_{p \in \mathbb{Z}} A^p \otimes B_{p+n}$$

  Recently, in an upcoming paper (CITE), the hom-complex was used to compute a parameterization for the the space of homotopy classes of chain maps between simplicial complexes.

- **Algebraic Mapping Cylinder:**

7 Examples

7.1 Simplicial Homology

In Figure 1, one can see an example of a filtered simplicial complex generated from points on a torus. As one moves from left to right, the filtration parameter, r, is increased yielding a more connected complex. In Figure 2 we show the persistence barcodes for the same shape. Note that the significant intervals corresponding to homological features that last for a long time in the filtration.

7.2 Matlab Scripting Example - Cellular Homology

In this section we show a brief Matlab session in which the cellular homology is computed for a Klein bottle over different coefficient fields. Essentially, this code examples constructs a cellular Klein bottle, initializes persistence algorithm objects over the fields $\mathbb{Z}/2\mathbb{Z}$, $\mathbb{Z}/3\mathbb{Z}$, and $\mathbb{Q}$, and computes the persistence intervals.

```
% get the cellular sphere of the specified dimension
stream = examples.CellStreamExamples.getCellularKleinBottle();

% get cellular homology algorithm over Z/2Z
Z2_persistence = api.Plex4.getModularCellularAlgorithm(3, 2);
% get cellular homology algorithm over Z/3Z
```

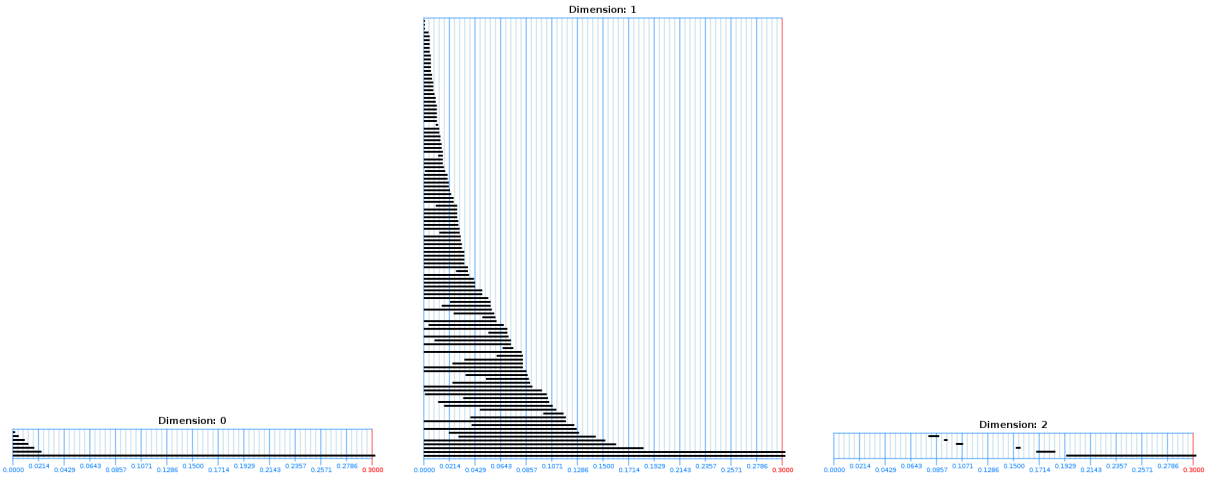

Figure 2: Persistence barcodes for a lazy-witness filtration of random points on a torus. The parameters used were: $N = 1000$, $L = 300$, $r_{\max} = 0.3$. The inner and outer radii of the torus were 0.5 and 1, respectively. The max-min selection procedure was used to create the landmark set. Note that long intervals correspond to significant homological features, and short ones are most likely the result of noise. We can see that the number of significant intervals in each dimension equals the expected Betti number.

```
Z3_persistence = api.Plex4.getModularCellularAlgorithm(3, 3);
% get cellular homology algorithm over Q
Q_persistence = api.Plex4.getRationalCellularAlgorithm(3);

% compute over Z/2Z - should give (1, 2, 1)
Z2_intervals = Z2_persistence.computeIntervals(stream)

% compute over Z/3Z - should give (1, 1, 0)
Z3_intervals = Z3_persistence.computeIntervals(stream)

% compute over Q - should give (1, 1, 0)
Q_intervals = Q_persistence.computeIntervals(stream)
```

The output of this example is:

```
Z2_intervals =
Dimension: 2 [0, infinity)
Dimension: 1 [0, infinity), [0, infinity)
Dimension: 0 [0, infinity)

Z3_intervals =
Dimension: 1 [0, infinity)
Dimension: 0 [0, infinity)

Q_intervals =
Dimension: 1 [0, infinity)
Dimension: 0 [0, infinity)
```

This is exactly what we expect, due to the presence of 2-torsion in the Klein bottle.

References

[Car09] Gunnar Carlsson, *Topology and data*, Bulletin of the American Mathematical Society **46** (2009), no. 2, 255–308.

[dSC04] Vin de Silva and Gunnar Carlsson, *Topological estimation using witness complexes*, Eurographics Symposium on Point-Based Graphics (M. Alexa and S. Rusinkiewicz, eds.), The Eurographics Association, 2004.

[dSMVJ10] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson, *Dualities in persistent (co)homology*, Unpublished manuscript, 2010.

[ZC05] Afra Zomorodian and Gunnar Carlsson, *Computing persistent homology*, Discrete Comput. Geom **33** (2005), 249–274.

[Zom10] A. Zomorodian, *Fast construction of the Vietoris-Rips complex*, Computers & Graphics **34** (2010), no. 3, 263 – 271.