Mapper Pipeline

Matthew Wright and Lori Ziegelmeier

June 9, 2017

Goal:

Simplify and visualize high dimensional data sets by reducing data to a simplicial complex.

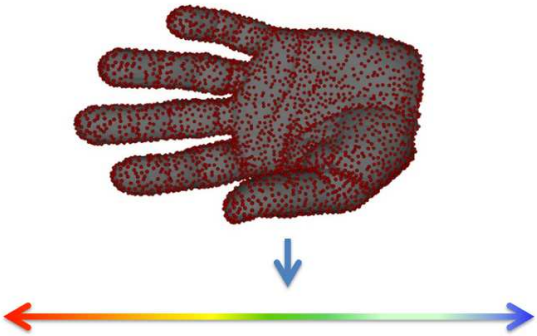
G. Singh, F. Memoli, G. Carlsson (2007). Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition, Point Based Graphics 2007, Prague, September 2007.

Pipeline

1. Take as input a high dimensional data set Y. (Need to be able to pairwise compute a measure of dissimilarity between points.)

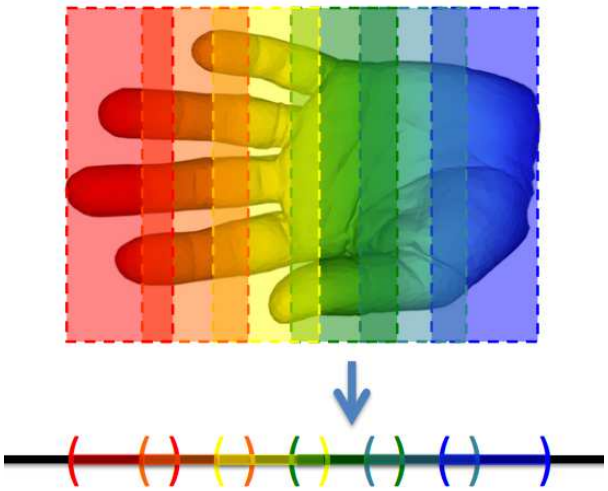
2. Define a filter function $f : Y \to \mathbb{R}$.

- Note the codomain could alternatively be $\mathbb{R}^n$ or the circle S^1.
- Example of the function could be projection onto one coordinate, a density estimator, eccentricity, magnitude, graph Laplacian, etc.



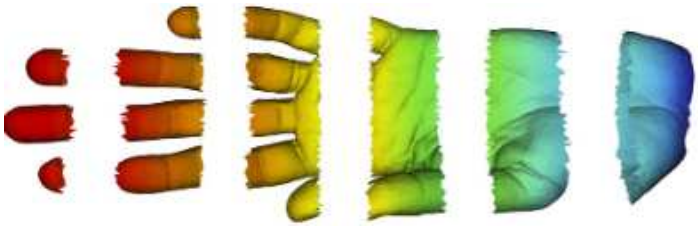
3. Put data in codomain $\mathbb{R}$ into overlapping bins and look at the preimage in the original data set, $f^{-1}(a_i, b_i) \in Y$.

- Choose length of intervals in $\mathbb{R}$ (if of equal length)
- Choose percentage of overlap



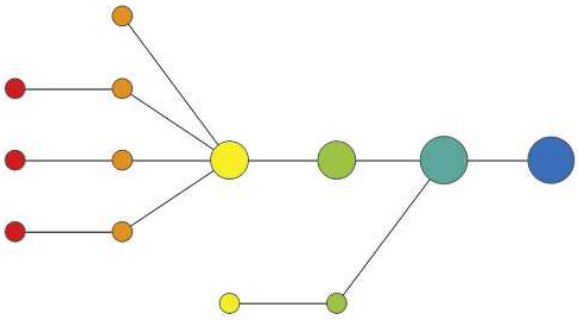
4. Cluster each preimage bin. Method of clustering not specified in original paper but do desire an algorithm:

- That use inter-point distance computations.
- That does not require the user to specify the number of clusters beforehand.
- For example, single linkage clustering could be used. This algorithm returns a vector which holds the length of the edge which was added to reduce the number of clusters by one at each step in the algorithm. The heuristic assumes that inter-point distance within a cluster is less than the distance between distinct clusters.



5. Create a simplicial complex from the clusters.

- Vertex = a cluster within a bin of a preimage
- Edge = nonempty intersection between clusters from different bins (exists because of overlapping intervals in $\mathbb{R}$).



Code Examples

To perform compuations we will use the TDAmapper package described here (https://github.com/paultpearson/TDAmapper/blob/master/README.md). The examples below come from that link.

First, we need to install the package

```
require(TDAmapper)
```

```
## Loading required package: TDAmapper
```

We will first use the mapper1D function within the TDAmapper package. This function takes as input

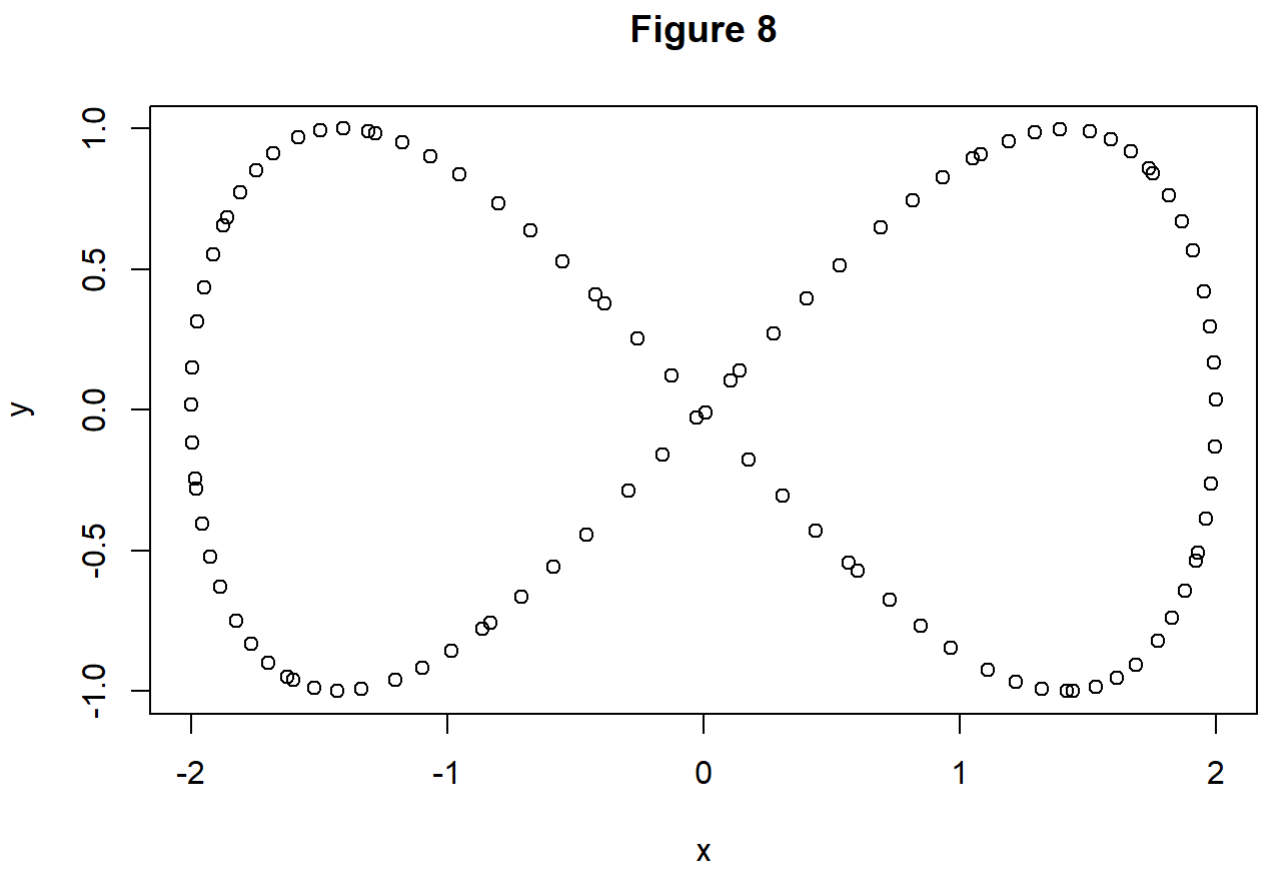
- distance_matrix: An nxn matrix of pairwise dissimilarities.
- filter_values: A length n vector of real numbers corresponding to the filter function on the data.
- num_intervals: A positive integer to divide $\mathbb{R}$ into.
- percent_overlap: A number between 0 and 00 specifying how much adjacent intervals should overlap.
- num_bins_when_clustering: A positive integer that controls whether points in the same preimage end up in the same cluster.

Now, we are ready to try our first example.

Example 1: Using mapper 1D to identify a figure 8

First, we create our dataset and plot it.

```
Figure8<-data.frame( x=2*cos(0.5*(1:100)), y=sin(1:100) )
plot(Figure8[,1],Figure8[,2],xlab='x',ylab='y',main='Figure 8')
```



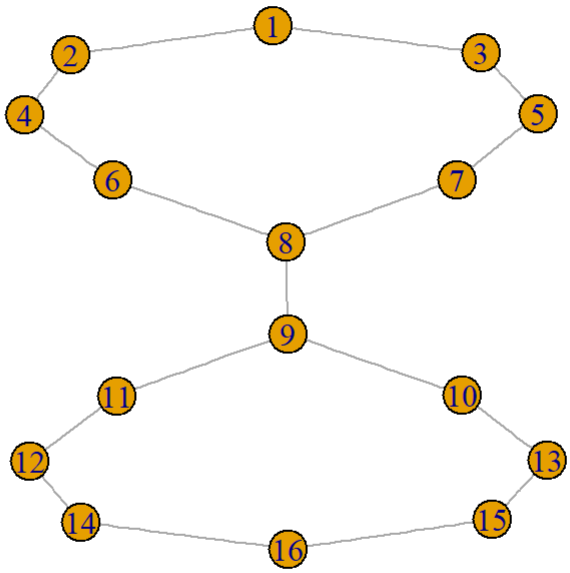
OK it is in fact a figure 8, but now we will forget that information and use mapper on the dataset.

```
# The fastcluster package is not necessary.  By loading the
# fastcluster package, the fastcluster::hclust() function
# automatically replaces the slower stats::hclust() function
# whenever hclust() is called.
require(fastcluster)

m1 <- mapper1D(
    distance_matrix = dist(Figure8),
    filter_values = Figure8[,1], #This defines the filter function to be the x value
    num_intervals = 10,
    percent_overlap = 50,
    num_bins_when_clustering = 10)

# The igraph package is necessary to view simplicial complexes
# (undirected graph) resulting from mapper1D().
require(igraph)

g1 <- graph.adjacency(m1$adjacency, mode="undirected")
plot(g1, layout = layout.auto(g1) )
```



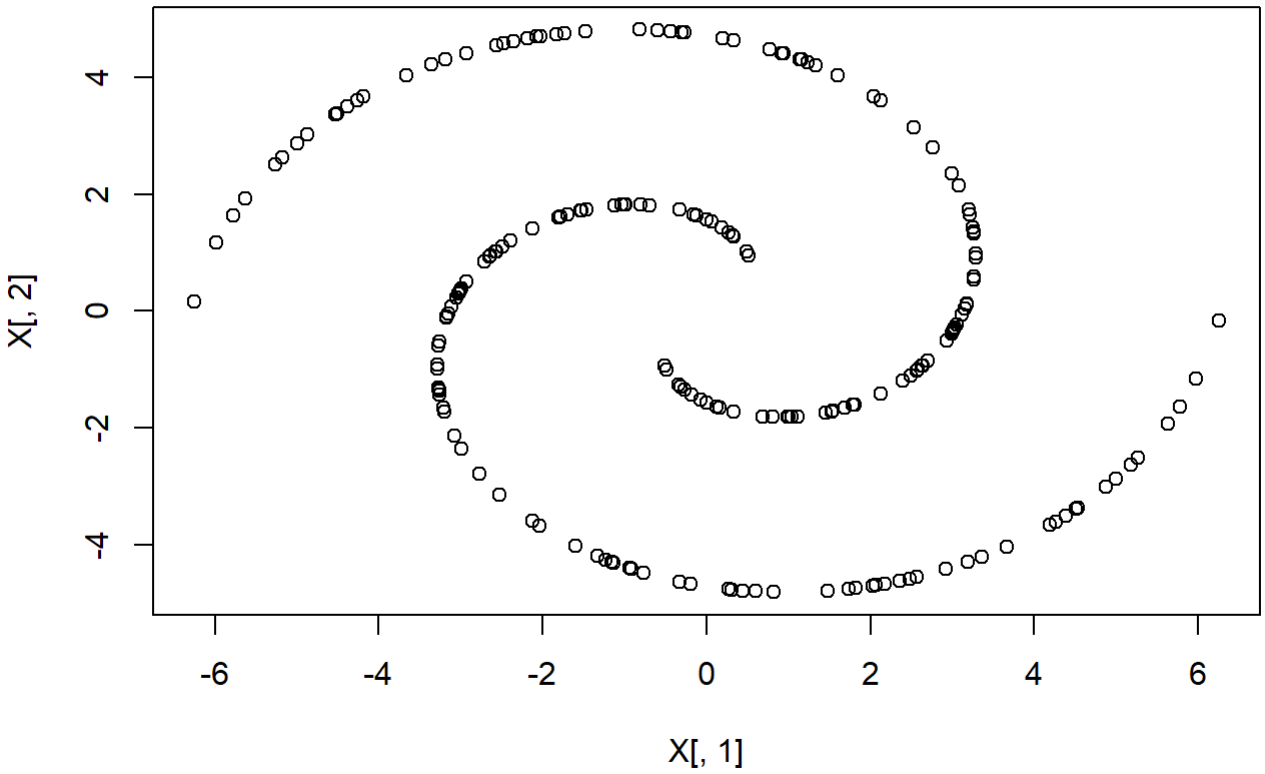
Play around with the parameters to see how the simplicial complex changes.

This dataset is in 2D, so we can actually visualize it without doing mapper. The real power of the algorithm is

when the data are high dimensional.

Example 2: Using mapper1D to identify two independent spirals as two line segments

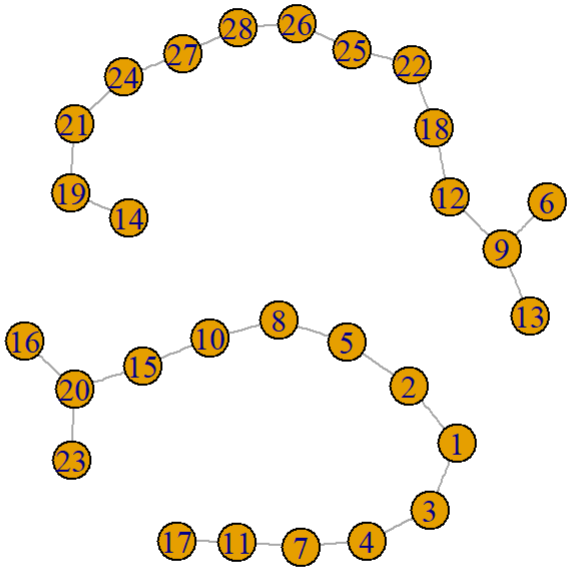
```
# sample points from two intertwined spirals
set.seed("1")
t <- runif(100, min=1, max=6.3) # theta
X <- data.frame( x = c( t*cos(t), -t*cos(t) ), y = c( t*sin(t), -t*sin(t) ) )
d <- dist(X)
plot(X[,1], X[,2])
```



```
filter <- X[,2] # height projection
num_intervals <- 10
percent_overlap <- 50
num_bins_when_clustering <- 10

m2 <- mapper1D(
    distance_matrix = d,
    filter_values = filter,
    # num_intervals = 10, # use default
    # percent_overlap = 50, # use default
    # num_bins_when_clustering = 10 # use default
    )

g2 <- graph.adjacency(m2$adjacency, mode="undirected")
plot(g2, layout = layout.auto(g2) )
```



Example 3: Using a new filter function to identify an oval as the circle S^1

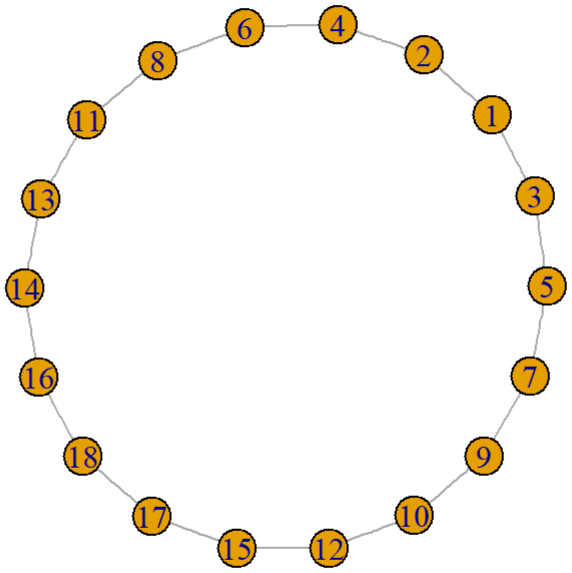
Now, let's consider a different filter function on the circle defined as the Euclidean distance from every point to a specified point within the dataset, i.e.

$$f : Y \to \mathbb{R}$$

$$f(y) = ||y - p||_2$$

```
circle <-cbind(2*cos(1:100), y=sin(1:100) )
circleSub <- circle-t(replicate(100,circle[1,])) #Subtract first point from all other
s
filter<-apply(circleSub,1,function(x){sqrt(sum(x^2))})
m3 <- mapper1D(
    distance_matrix = dist(circle),
    filter_values = filter,
    num_intervals = 10,
    percent_overlap = 50,
    num_bins_when_clustering = 10)

g3 <- graph.adjacency(m3$adjacency, mode="undirected")
plot(g3, layout = layout.auto(g3) )
```



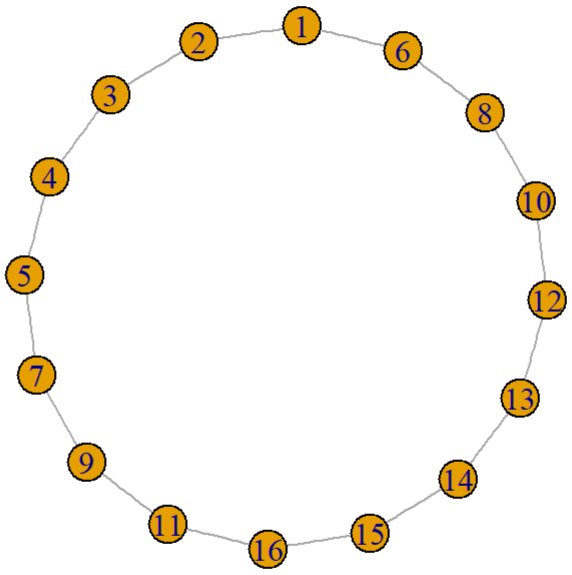
Example 4: Using mapper2D to identify an oval as the circle S^1

The codomain of the filter function can be spaces other than $\mathbb{R}$. For instance, one may define the filter function $f : Y \to \mathbb{R}^2$ specified by the parameter - filter_values: a list of two length n vectors of real numbers. - num_intervals: a vector of two positive integers

```
m4 <- mapper2D(
    distance_matrix = dist(data.frame( x=2*cos(1:100), y=sin(1:100) )),
    filter_values = list( 2*cos(1:100), sin(1:100) ),
    num_intervals = c(5,5),
    percent_overlap = 50,
    num_bins_when_clustering = 10)
```

```
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
## [1] "Level set is empty"
```

```
g4 <- graph.adjacency(m4$adjacency, mode="undirected")
plot(g4, layout = layout.auto(g4) )
```



Exploration

Now, it's your turn to do some exploration!

There are two additional examples to be found at the github TDAmapper link (https://github.com/paultpearson/TDAmapper/blob/master/README.md) that are interactive. We encourage you to try them out.

There are several datasets available on our conference website (http://pages.stolaf.edu/tda-conference/program/data/). Use mapper with a variety of filter functions and choices of parameters to see if you can discover topological structure.