PERSNET TUTORIAL
METRICS AND PERSISTENCE ON NETWORKS

SAMIR CHOWDHURY AND FACUNDO MÉMOLI

ABSTRACT. This is a short tutorial to accompany the `PersNet` package. The package and associated papers can be obtained from https://research.math.osu.edu/networks/.

CONTENTS

1. Installation instructions 1
2. Metrics on Networks 2
2.1. Random networks 2
2.2. Cycle networks 2
2.3. Cycle metric graphs 4
2.4. Computing a lower bound based on local spectrum 4
2.5. Application: Community detection in random graphs 6
2.6. Application: Hippocampal networks database 6
3. Persistence on Networks 7
3.1. Computing a Rips filtration 8
3.2. Computing a Dowker filtration 8
3.3. Application: Persistence based clustering on hippocampal networks database 11
4. Further applications 11
4.1. The `econsmall` database 11
4.2. The `econfull` database 12
4.3. The `usmigration` data 12
4.4. Challenge: data in the wild 13
References 13

1. INSTALLATION INSTRUCTIONS

You may download the `PersNet` package from https://research.math.osu.edu/networks/Datasets.html. After unzipping the contents of the folder, you may navigate to the `persnet` directory in Matlab and type in the following:

```
>>addpath(genpath(pwd))
```

Additionally, you will need to download and install `javaplex` [3] to use all the features in this package. This software can be obtained from http://appliedtopology.github.io/

E-mail address: chowdhury.57@osu.edu, memoli@math.osu.edu.
Date: January 19, 2017.

javaplex/ (last accessed Jan 17, 2017). Follow the installation instructions in the associated documentation.

2. Metrics on Networks

In this section, we will generate a few networks and compute their network distances. Then we will see how to use the local spectrum bound to obtain approximations of the network distance.

2.1. **Random networks.** Random networks are easy to generate. As a first step, navigate to the `persnet` folder in Matlab and type in the following to set the working directory:

```
>> addpath(genpath(pwd))
```

Next type the following into Matlab to compute two random networks wX, wY and their network distance $d_{\mathcal{N}}(wX, wY)$.

```
>> wX=randi(10,3,3)

wX =

     4     8     1
     7     1     1
     2     3     9

>> wY=randi(10,3,3)

wY =

     7     1     8
     4     5     8
    10     4     2

>> computeDN(wX,wY)

ans =

    2.5000
```

Exercise 1. Repeat the preceding calculation after taking wY to be: (1) a negative matrix, (2) a matrix with decimal values, and (3) a 4×4 matrix.

Exercise 2. Computationally test the following conjectures:

(1) For a two node network wX, $d_{\mathcal{N}}(wX, wX') = 0$. Here wX' denotes the transpose of wX.
(2) For a three node network wX, in general $d_{\mathcal{N}}(wX, wX') \neq 0$.

Computing the network distance is generally a difficult problem. With the `computeDN` code provided in this package, it is not recommended to compute network distances for networks containing more than four nodes.

2.2. **Cycle networks.** Let us now produce some asymmetric networks that we understand better. The `cycleNetwork` code can be used to compute *cycle networks*. There are two input arguments: the first controls the number of nodes, and the second controls the "forward" weight.

```
>> wX=cycleNetwork(5,1)

wX =

     0     1     2     3     4
     4     0     1     2     3
     3     4     0     1     2
     2     3     4     0     1
     1     2     3     4     0

>> wY=0

wY =

     0

>> computeDN(wX,wY)

ans =

     2
```

In the above, notice that there is a unique correspondence between the cycle network and the one-point network, and the distortion of this correspondence is equal to the diameter of the cycle network.

Exercise 3. Compute a 4×4 table of network distances between cycle networks on n and m nodes, $1 \leq n, m \leq 4$, respectively. Set the forward weight on each cycle network equal to 1. A similar computation is shown below.

```
>> wX=cycleNetwork(5,1)

wX =

     0     1     2     3     4
     4     0     1     2     3
     3     4     0     1     2
     2     3     4     0     1
     1     2     3     4     0

>> wY=cycleNetwork(3,1)

wY =
```

```
     0     1     2
     2     0     1
     1     2     0

>> computeDN(wX,wY)

ans =

     1
```

Often we will be interested in the *metric* version of a cycle network. This is described in the next section.

2.3. **Cycle metric graphs.** The `cycleMetricGraph` code can produce a *metric* cycle graph. A simple computation is shown below. Notice that when comparing metric spaces, the network distance is equivalent to the Gromov-Hausdorff distance.

```
>> wX=cycleMetricGraph(5,1)

wX =

     0     1     2     2     1
     1     0     1     2     2
     2     1     0     1     2
     2     2     1     0     1
     1     2     2     1     0

>> wY=cycleMetricGraph(3,1)

wY =

     0     1     1
     1     0     1
     1     1     0

>> computeDN(wX,wY)

ans =

    0.5000
```

So far, we have been restricted to computing network distances between very small networks. In the next section, we remove this obstruction by using lower bounds based on the local spectrum.

2.4. **Computing a lower bound based on local spectrum.** The `matchLocSpec` code can be used to compute a lower bound on the distance between two networks using local spectra. Notice that this is much faster than computing the network distance!

```
>> tic
wX=cycleMetricGraph(5,1)
```

```
wY=cycleMetricGraph(3,1)
computeDN(wX,wY)
toc

wX =

     0     1     2     2     1
     1     0     1     2     2
     2     1     0     1     2
     2     2     1     0     1
     1     2     2     1     0


wY =

     0     1     1
     1     0     1
     1     1     0


ans =

    0.5000

Elapsed time is 3.269978 seconds.
>> tic
wX=cycleMetricGraph(5,1)
wY=cycleMetricGraph(3,1)
[~,c]=matchLocSpec(wX,wY)
toc

wX =

     0     1     2     2     1
     1     0     1     2     2
     2     1     0     1     2
     2     2     1     0     1
     1     2     2     1     0


wY =

     0     1     1
     1     0     1
     1     1     0
```

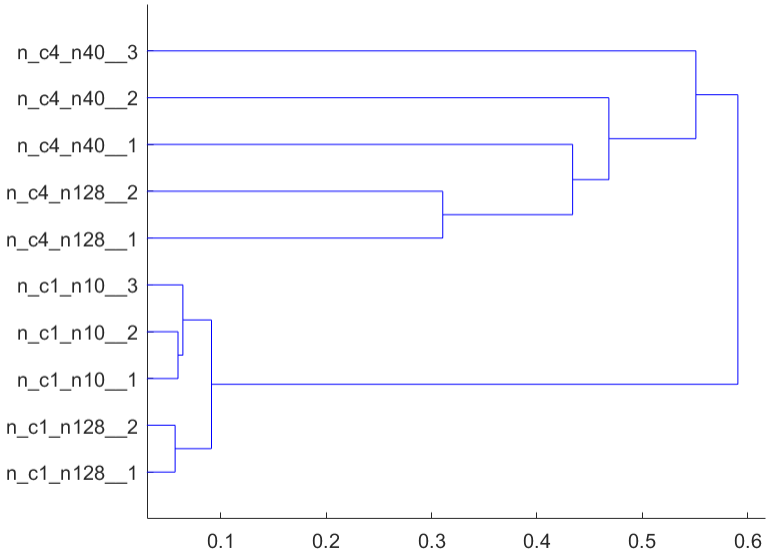

FIGURE 1. Dendrogram obtained from running `demoLocSpecFor`.

```
c =

     1

Elapsed time is 0.049823 seconds.
```

Exercise 4. Compute a 4×4 table of local spectra lower bounds for cycle networks on n and m nodes, $1 \leq n, m \leq 4$, respectively. Set the forward weight on each cycle network equal to 1. Compare the results with those obtained in Exercise 3.

2.5. **Application: Community detection in random graphs.** As a demonstration of the capabilities of the local spectrum lower bounds, we apply the `matchLocSpec` to a database of 10 random networks generated from software written by Santo Fortunato [1]. The database contains 3 networks of four communities with 40 nodes, 2 networks of four communities with 128 nodes, 3 networks of one community with 10 nodes, and 2 networks of one community with 128 nodes.

Run the Matlab script file `demoLocSpecFor`. The results will be computed in a few minutes. The resulting dendrogram is presented in Figure 1.

2.6. **Application: Hippocampal networks database.** The folder `Hippo25` contains a collection of 25 *hippocampal networks*. Briefly, a hippocampal network records the firing pattern of hippocampal cells in the brain of a rodent as it navigates an arena with a number of obstacles. The 25 networks in this collection are obtained from simulations of a rodent's movement across arenas containing 0, 1, 2, 3 and 4 obstacles. Full details of how these networks are obtained can be found in [2].

To gain a better understanding of these networks, type the following (from the `persnet` directory):

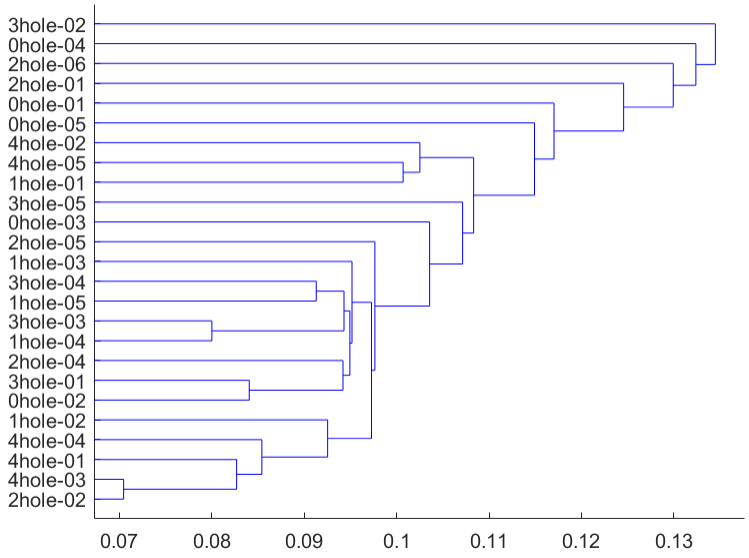

FIGURE 2. Dendrogram obtained from running `demoLocSpecHippo`. Compare this to the dendrogram in Figure 5.

```
>> addpath(genpath(pwd))
>> cd Hippo25\4hole\
>> load t_pcells01.mat;
>> load t_raster01.mat;
>> testRaster(cells,firingRaster)
```

The dots on the screen represent physical locations that different place cells associate themselves with. As the rodent enters a physical region, the associated place cell begins to fire. Notice that there are four "obstacles," i.e. regions that the rodent cannot access.

The ground truth of each network provided in the folder `allDissim` is the number of obstacles in the corresponding arena. In the next section, we will see how this can be done using persistence. However, we can still attempt to perform a clustering task on these networks using the local spectrum lower bounds.

Return to the `persnet` folder and run the `demoLocSpecHippo` code. Warning: this task can take several hours to complete! For convenience, all the results have been precomputed and placed in the appropriate folder. Matlab will simply plot the resulting dendrogram, which we have presented in Figure 2.

3. Persistence on Networks

We will now implement code for computing Rips and Dowker persistence on networks. As a prerequisite, you need to have `javaplex` installed [3]. This software can be obtained from http://appliedtopology.github.io/javaplex/ (last accessed Jan 17, 2017).

The `javaplex` installation folder contains a file called `load_javaplex`. You will need to run this file each time you start Matlab in order to implement the code presented in this section. An easy test to help determine if Matlab is loading the necessary files is to type the following:

```
>> which plot_barcodes
```

Matlab should return a valid directory if things are working correctly.

A note on integer and non-integer valued networks. Because of the way `javaplex` works, one needs to supply the diameter whenever computing persistence of a non-integer valued network. This will be illustrated in the next section.

3.1. **Computing a Rips filtration.** As a first example, we compute the Rips persistence barcodes in dimensions 1 and 2 of a cycle network. The barcodes are presented in Figure 3.

```
>> A=cycleNetwork(7,1)

A =

     0     1     2     3     4     5     6
     6     0     1     2     3     4     5
     5     6     0     1     2     3     4
     4     5     6     0     1     2     3
     3     4     5     6     0     1     2
     2     3     4     5     6     0     1
     1     2     3     4     5     6     0

>> [sk0,sk1,sk2,~]= rips(A);
>> computePers(sk0,sk1,sk2,[])

represent =

Dimension: 0
[0.0, 4.0): [4] + -[1]
[0.0, 4.0): [5] + -[1]
[0.0, 4.0): -[2] + [1]
[0.0, 4.0): [6] + -[2]
[0.0, 4.0): [2] + -[3]
[0.0, 4.0): [7] + -[3]
[0.0, infinity): [1]
Dimension: 1
[4.0, 5.0): [2,6] + [3,6] + [4,7] + [1,4] + [1,5] + [2,5] + -[3,7]
[5.0, 6.0): [3,6] + [1,3] + [2,6] + -[1,5] + [2,5]
```

Exercise 5. Compute the Rips persistence barcodes of cycle networks having n nodes for $3 \leq n \leq 20$. Set the forward weight equal to 1. Do you see any discernible pattern arising?

3.2. **Computing a Dowker filtration.** We now repeat the same computation as in the preceding section, but this time, we compute the Dowker persistence instead of Rips persistence. The resulting barcode is illustrated in Figure 4.

```
>> A=cycleNetwork(7,1)

A =
```

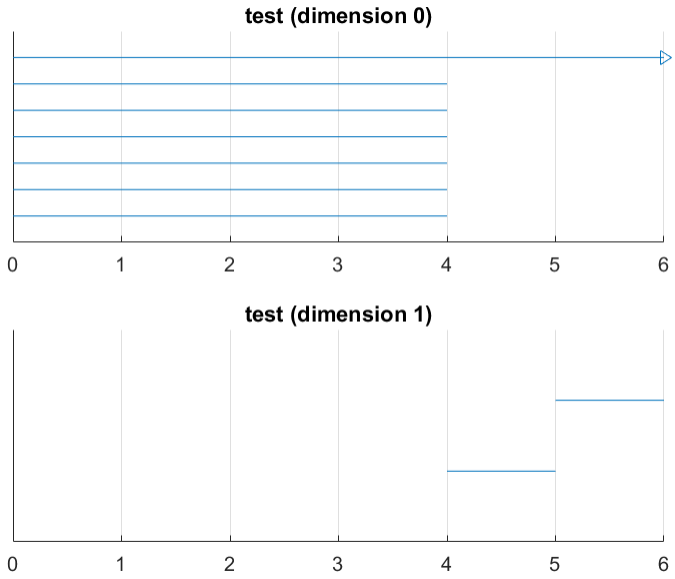

FIGURE 3. Rips persistence barcodes of a cycle network on 7 nodes.

```
     0     1     2     3     4     5     6
     6     0     1     2     3     4     5
     5     6     0     1     2     3     4
     4     5     6     0     1     2     3
     3     4     5     6     0     1     2
     2     3     4     5     6     0     1
     1     2     3     4     5     6     0

>> [sk0,sk1,sk2,~]= dowker(A);
>> computePers(sk0,sk1,sk2,[])

represent =

Dimension: 0
[0.0, 1.0): [2] + -[1]
[0.0, 1.0): [7] + -[1]
[0.0, 1.0): -[2] + [3]
[0.0, 1.0): [4] + -[3]
[0.0, 1.0): -[4] + [5]
[0.0, 1.0): [6] + -[5]
[0.0, infinity): [1]
Dimension: 1
[1.0, 4.0): [4,5] + [1,7] + [2,3] + [5,6] + [6,7] + [1,2] + [3,4]
```

Exercise 6. Compute the Dowker persistence barcodes of cycle networks having n nodes for $3 \leq n \leq 20$. Set the forward weight equal to 1. Do you see any discernible pattern arising?

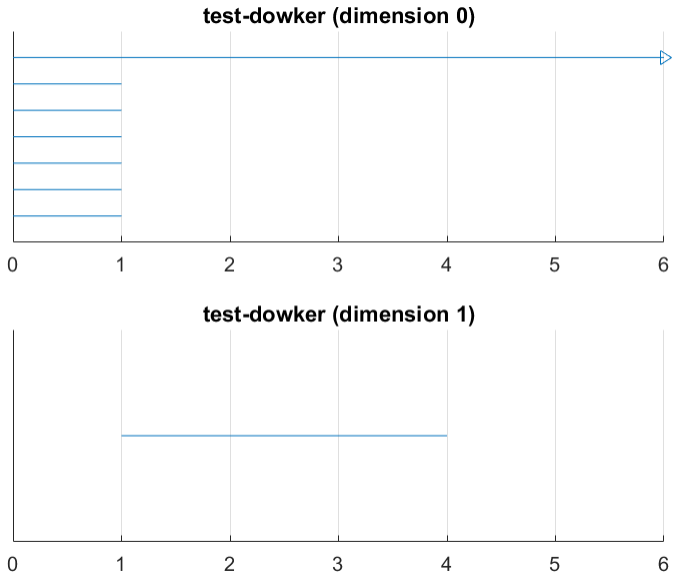

FIGURE 4. Dowker persistence barcodes of cycle network on 7 nodes.

Exercise 7 (Transposition invariance of Rips and Dowker)**.** Compute the Rips and Dowker persistence barcodes of a random network and its transpose, as in the next example:

```
>> wX=rand(7,7);
>> diam=max(max(wX));
>> [sk0,sk1,sk2]=rips(wX);
>> computePers(sk0,sk1,sk2,[],diam)

represent =

Dimension: 0
[0.38160384769526423, 0.44813156686284106): [6] + -[2]
[0.3330529408940852, 0.5725543197105271): [2] + [5]
[0.2875594435363843, 0.5943745054039881): [7] + [2]
[0.04124174077380656, infinity): [2]
Dimension: 1
[0.6068237361213548, 0.7824017084615934): [4,5] + [3,6] + [4,6] + [3,5]


>> [sk0,sk1,sk2]=rips(wX');
>> computePers(sk0,sk1,sk2,[],diam)

represent =

Dimension: 0
[0.38160384769526423, 0.44813156686284106): [6] + -[2]
[0.3330529408940852, 0.5725543197105271): [2] + [5]
```

```
[0.2875594435363843, 0.5943745054039881): [7] + [2]
[0.04124174077380656, infinity): [2]
Dimension: 1
[0.6068237361213548, 0.7824017084615934): [4,5] + [3,6] + [4,6] + [3,5]

---------------------------

>> [sk0,sk1,sk2]=dowker(wX);
>> computePers(sk0,sk1,sk2,[],diam)

represent =

Dimension: 0
[0.10535592482984348, 0.19131910994296208): -[3] + [5]
[0.06579441784509169, 0.20360141027797443): [4] + [5]
[0.0716588411585164, 0.20427409997353332): [6] + -[2]
[0.04124174077380656, 0.2575775545056697): [4] + [2]
[0.2875594435363843, 0.29090003844992784): [7] + -[3]
[0.015916017050904292, infinity): [4]

>> [sk0,sk1,sk2]=dowker(wX');
>> computePers(sk0,sk1,sk2,[],diam)

represent =

Dimension: 0
[0.10535592482984348, 0.19131910994296208): [6] + -[3]
[0.06579441784509169, 0.20360141027797443): [3] + -[5]
[0.0716588411585164, 0.20427409997353332): [4] + -[2]
[0.04124174077380656, 0.2575775545056697): -[2] + [5]
[0.2875594435363843, 0.29090003844992784): [7] + -[5]
[0.015916017050904292, infinity): [5]
```

Is it a coincidence that transposition has no effect on the Rips and Dowker persistence barcodes? Test this computationally by generating random networks having up to 20 nodes. Compare your answer to the preceding question with Proposition 18 of [2].

3.3. **Application: Persistence based clustering on hippocampal networks database.** We now compute Dowker persistence barcodes of the hippocampal networks in the `Hippo25` folder. For convenience, the skeleta have been precomputed. To run this demonstration, simply run `demoDowkerHippo` from the `persnet` folder. The resulting dendrogram is presented in Figure 5.

Results from a similar computation on a database of 100 networks is provided in [2]. You may also find the dissimilarity networks used in that computation in the `Hippo100` folder. Yet another set of results from computations on a database of 3000 networks is illustrated in https://research.math.osu.edu/networks/dowker/arenaVid.html.

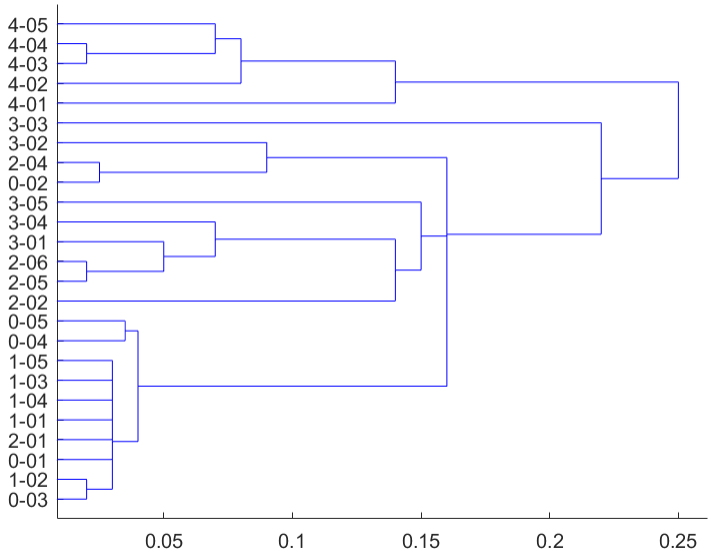

FIGURE 5. Single linkage dendrogram obtained by computing bottleneck distances between 1-dimensional Dowker persistence barcodes of hippocampal networks.

4. Further applications

4.1. **The `econsmall` database.** The `econsmall` folder contains a set of 19 networks representing the flow of commodities across 15 U.S. economic sectors over the years 1997-2015. Can the persistence-based methods presented in this package help find significant changes in the U.S. economy, perhaps those caused by the 2007-2008 financial crisis?

Run `demoLocSpecEcon` and `demoDowkerEcon` to see the dendrograms obtained by applying the local spectrum and Dowker methods to this dataset. Compare the results with those obtained in [4].

4.2. **The `econfull` database.** The `econfull` folder contains a network representing the flow of commodities across 71 U.S. economic sectors in the year 2011. To perform some exploratory data analysis on this network, type in the following:

```
>> load dissim.mat
>> [sk0,sk1,sk2]=dowker(dissim);
>> diam=max(max(dissim));
>> computePers(sk0,sk1,sk2,[],diam)
```

Note that `javaplex` returns generators of persistent 1-cycles. By comparing with the labels provided in `labels.mat`, one can know exactly which economic sectors generated persistent *loops* in the economy. What can you conjecture about the meaning of such a loop? Can you extract a meaningful story from some of the longest-persisting 1-cycles? Compare with the discussion in [2].

4.3. **The `usmigration` data.** The `usmigration` folder contains state-to-state migration data for 50 U.S. states, the District of Columbia, and Puerto Rico. The methods used above

can also be used to find generators for persistent 1-cycles in this dataset, which can then be identified using the provided labels.

What is the meaning of a persistent 1-cycle in this setting? Compare your answer with the discussion in [2].

4.4. **Challenge: data in the wild.** The method described in this tutorial can be applied to a wide range of datasets. Some places to look for data include the Latin America Migration Project, the World Bank databases, the Stanford Large Network Dataset Collection, and the UCI Network Data Repository.

Pick a dataset you care about and analyze it using any of the methods presented in this package. You may be surprised by the story contained in the data!

References

[1] S. Fortunato, "Benchmark graphs to test community detection algorithms." https://sites.google.com/site/santofortunato/inthepress2.

[2] S. Chowdhury and F. Mémoli, "Persistent homology of asymmetric networks: An approach based on dowker filtrations," *arXiv preprint arXiv:1608.05432*, 2016.

[3] A. Tausz, M. Vejdemo-Johansson, and H. Adams, "Javaplex: A research software package for persistent (co) homology,"

[4] S. Chowdhury and F. Mémoli, "Persistent path homology of directed networks," *arXiv preprint arXiv:1701.00565*, 2017.