Complex Neuronal Contagions

Release version 1.0, 2020

Bengier Ülgen Kılıç

May 01, 2021

CONTENTS:

1 Introduction **3**
- 1.1 Geometric and Noisy Geometric Networks . . . 3
- 1.2 Contagion Model . . . 3

2 Quick Tutorial **5**
- 2.1 Installation/Usage . . . 5
- 2.2 Initiate a `geometric_network` object . . . 5
- 2.3 Add noise to geometric network . . . 5
- 2.4 Display the network via Networkx . . . 5
- 2.5 Sample Excitation Simulation . . . 6
- 2.6 Look at the first activation times . . . 7
- 2.7 Create Distance Matrix . . . 8
- 2.8 Contagion Size . . . 9
- 2.9 Persistence Diagrams . . . 10

3 The `geometric_network` class **13**

4 Indices and tables **19**

Index **21**

Complex Contagions is a python package to run several different contagion models on custom networks to understand and analyze the dynamics of these systems.

CHAPTER

ONE

INTRODUCTION

`Complex_Contagions` is a python module to run our experiments. In this contagion model, we want to investigate the dynamics of a contagion starting from a seed cluster and spreading across the underlying network. The model and hence the package is as general as possible in a way that one can play with the parameters to obtain different network topologies and contagion models.

1.1 Geometric and Noisy Geometric Networks

A geometric network is a set of nodes and edges where the nodes connected to their 'close' neighbors in a euclidean distance manner.

Noisy geometric networks are obtained by adding 'noise' or edges that connects 'distant' nodes to the geometric networks. These network topology manipulations are shown to be demonstrated various contagion spread phenomenans such as wavefront propagation(WFP) or appearance of new clusters(ANC) in these networks.

1.2 Contagion Model

We are inspired by a neuoronal contagion model to asses this two phenomenans. The core function that we run our experiments decides if a given neuron is going to fire or not by a sigmoid function $$f(x) = /frac{1}{1+exp^{-C.x}} $$. The main class we use `geometric_network` comes with several methods that we can manipulate the nature of the contagion very easily. For example, one can run either a stochastic or deterministic model by varying the parameter C or users have the option to choose if neuorns are going to have a refractory period that they are not allowed to fire right after a spike.

CHAPTER

TWO

QUICK TUTORIAL

2.1 Installation/Usage

As the package has not been published on PyPi yet, it CANNOT be install using pip.

For now, the suggested method is to put the file *Complex_Contagions.py* in the same directory as your source files and call `from Complex_Contagions import geometric_network`

2.2 Initiate a `geometric_network` object

Create a geometric network on a ring. `Band_length` corresponds to the number of neighbors to connect from both right and left making the geometric degree 2*band_length

```
n = 20
d2 = 2
ring_latt=  geometric_network('ring_lattice', size = n, banded = True, band_length =␣
↪3)
```

2.3 Add noise to geometric network

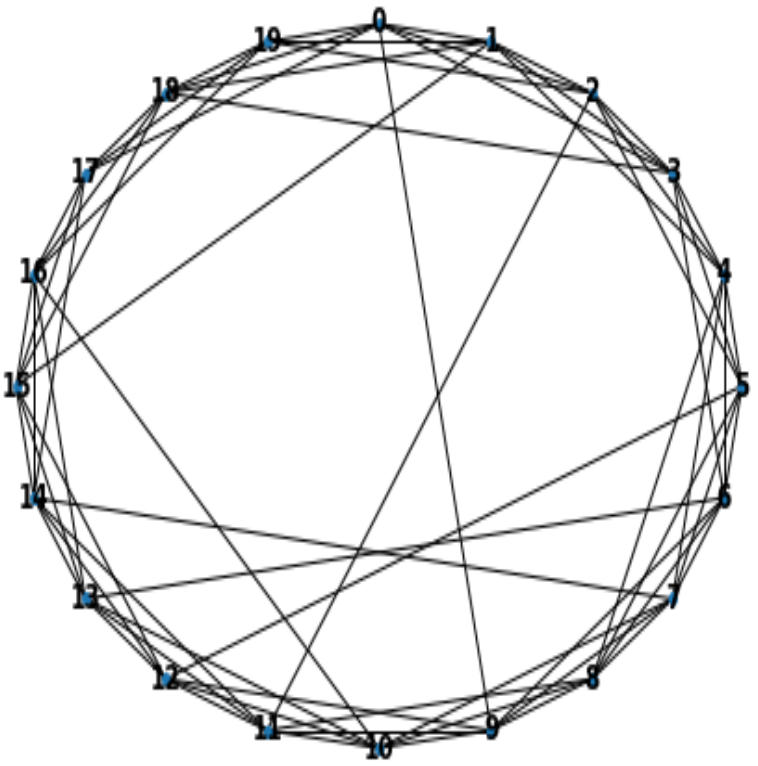
Use `add_noise_to_geometric()` method to manipulate the network topology. The second parameter describes the non-geometric degree of every node.

```
ring_latt_k_regular.add_noise_to_geometric('k_regular', d2)
```

2.4 Display the network via Networkx

Spy the network.

```
ring_latt_k_regular.display()
```



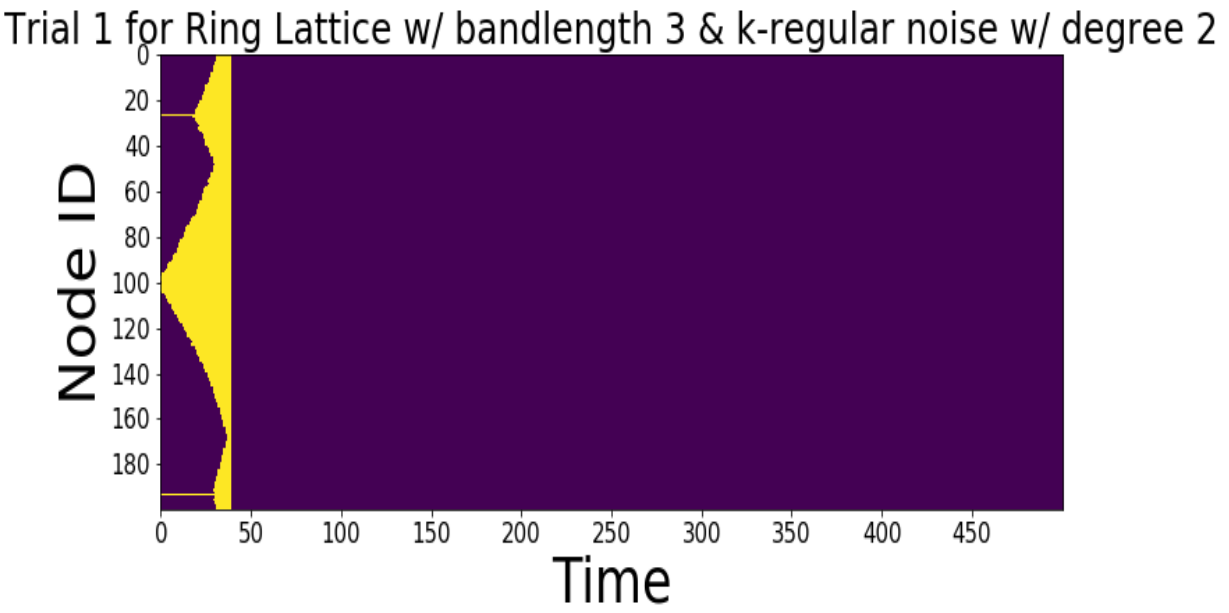
2.5 Sample Excitation Simulation

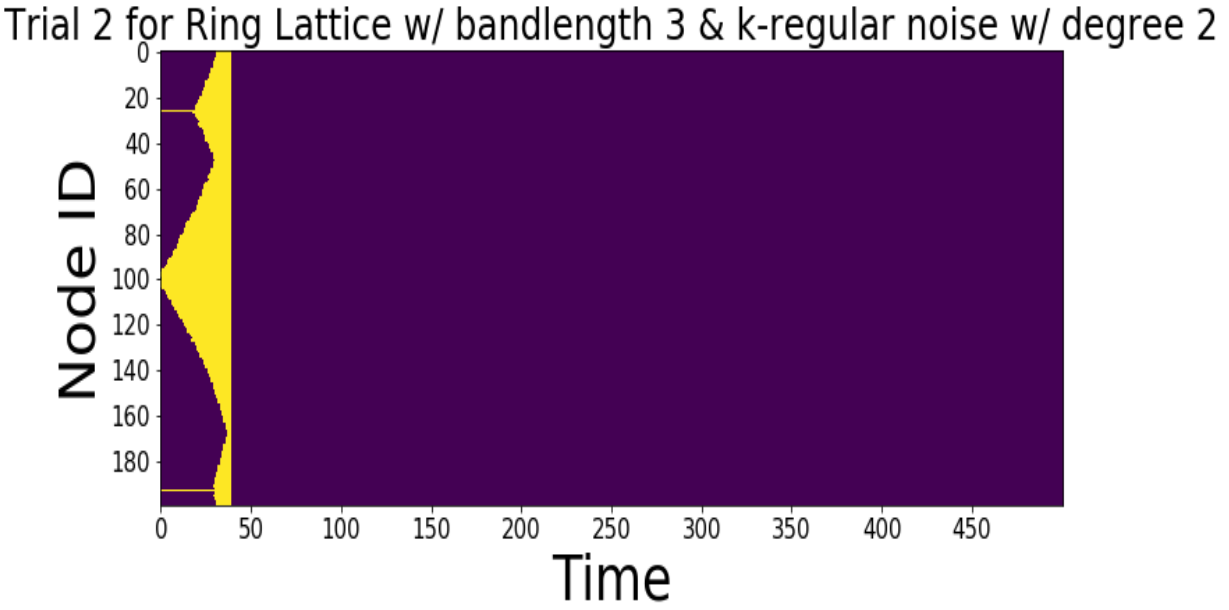
Run the complex contagion on the network we have created. Key parameters are threshold, C and $alpha = frac{nGD}{GD}$

```
n = 200
d2 = 2
ring_latt_k_regular =  geometric_network('ring_lattice', size = n, banded = True,␣
↪band_length = 3)
ring_latt_k_regular.add_noise_to_geometric('k_regular', d2)

T = 100 # number of iterations
seed = int(n/2) # node that the spread starts
C = 1000 # Geometrically, this describes the turning of the sigmoid function
threshold = 0.3 # resistence of the node to it's neighbors' excitation level
Trials = 2 # number of trials
refractory_period = False ## if a neuron is activated once, it stays activated␣
↪throughout.

fig, ax = plt.subplots(Trials,1, figsize = (50,10))
first_excitation_times, contagion_size = ring_latt_k_regular.run_excitation(Trials, T,
↪ C, seed, threshold, refractory_period, ax = ax)
```

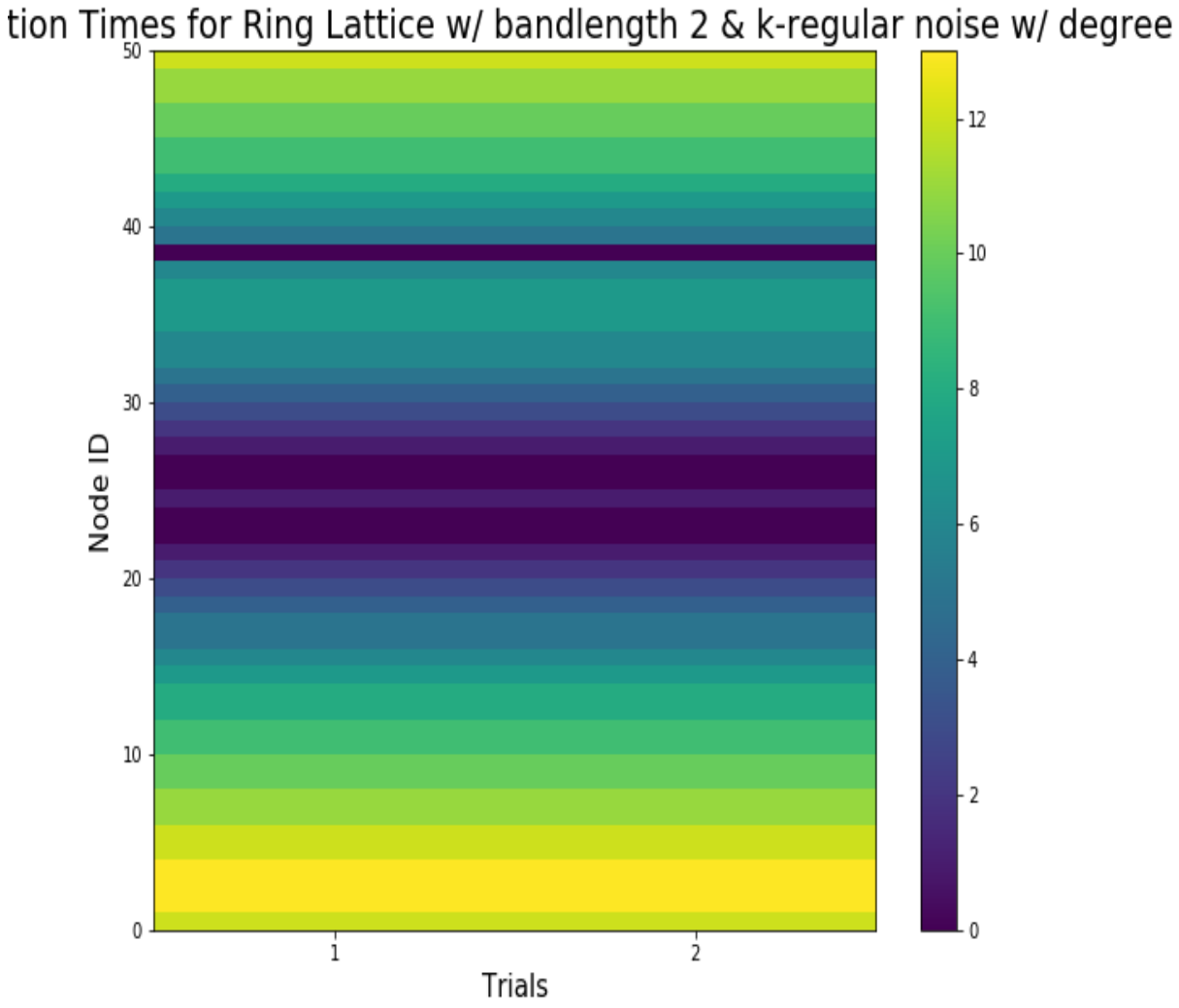




2.6 Look at the first activation times

Spy activation of the nodes.

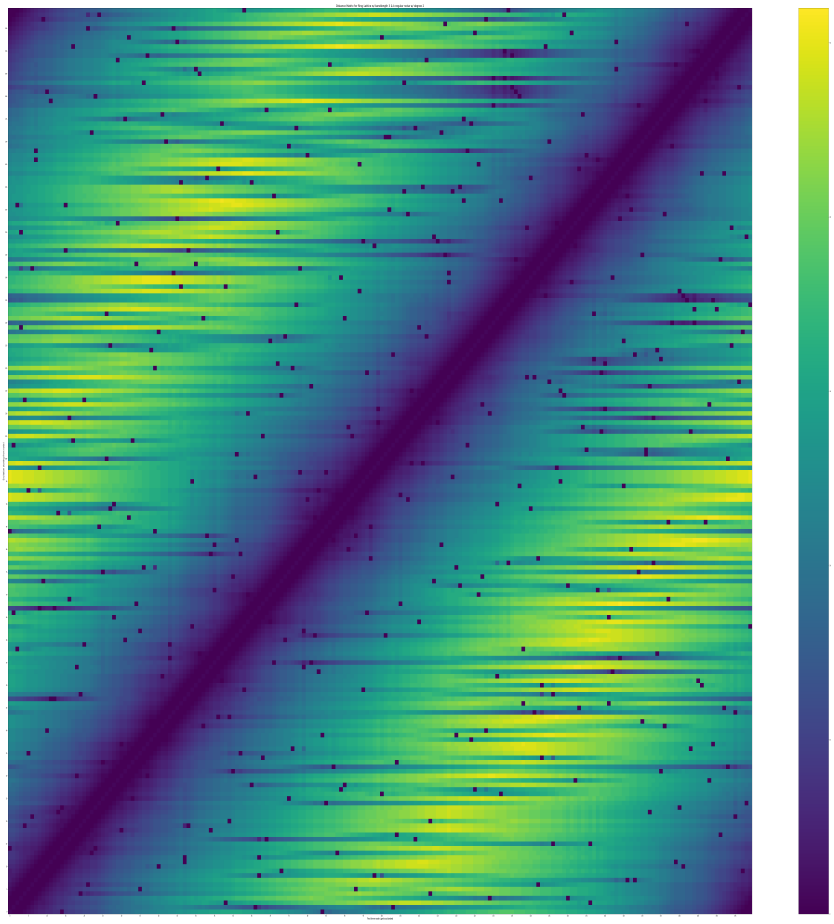
```
ring_latt_k_regular.spy_first_activation(first_excitation_times)
```



2.7 Create Distance Matrix

If you don't need to look at the individual contagions starting from different nodes, you can run the contagion starting from node i and calculating the first time it reaches to node j i.e. create a distance matrix who (i,j) entry is the first time the node j activated on a contagion starting from i.

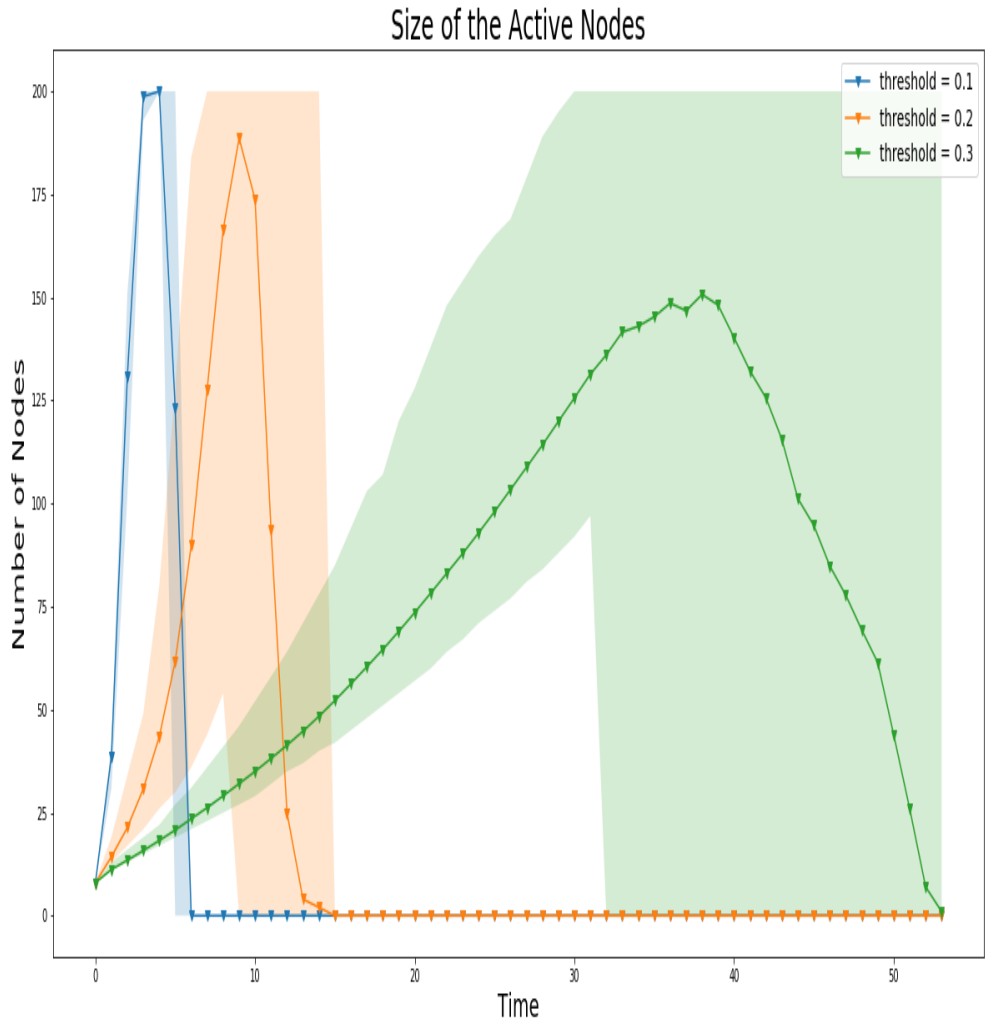
```
D, Q = ring_latt_k_regular.make_distance_matrix(T, C, threshold, Trials, refractory_
↪period, spy_distance = True)
```



2.8 Contagion Size

It's important to look at the bifurcations in the system. In order to do so, one might need to look at the size of the contagion for example different thresholds.

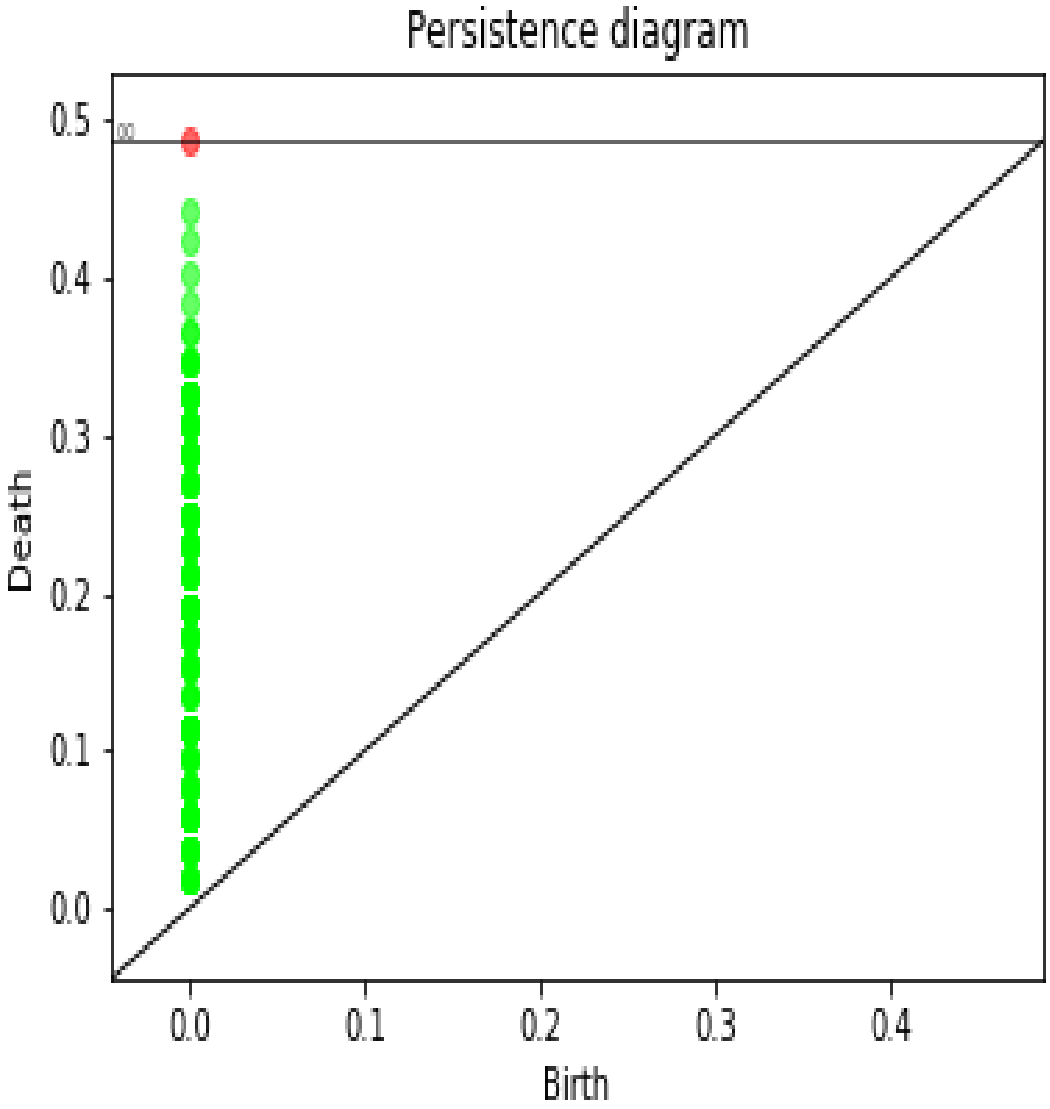
```
labels = ['threshold = 0.1', 'threshold = 0.2', 'threshold = 0.3']
Q = [Q1,Q2,Q3]  ## Qi is the second output of the ``make_distance_matrix``
ring_latt_k_regular.display_comm_sizes(Q,labels)
```



2.9 Persistence Diagrams

Once we created the distance matrices, we can look at the topological features across different contagions and different topologies.

```
pers = ring_latt_k_regular.compute_persistence(D, spy = True)
delta = ring_latt_k_regular.one_d_Delta(pers)
```


Persistence diagram
Death
Birth
0.0
0.1
0.2
0.3
0.4
0.5
∞

CHAPTER

THREE

THE GEOMETRIC_NETWORK CLASS

class Complex_Contagions.**geometric_network**(*network_type*, *size*, ***kwargs*)

Bases: object

Geometric Network object to run complex contagions on.

N

Size, number of nodes in the network.

Type int

M

Total number of edges in the network.

Type int

graph

Networkx graph corresponding to the geometric_network object. You can use all the networkx library with this attribute.

Type a Networkx object

pos

A dictionary of nodes and their spatial location.

Type dict

A

Adjacency matrix of the graph. Use .todense() or .toarray() to manipulate.

Type A Scipy sparse matrix

text

a simple description of the network.

Type str

Parameters

- **network_type** (*str*) – Type of the network to be created. It can be 2D_lattice or 'ring_lattice``.
- **size** (*int*) – Size of the network to be initated. If 2D_lattice, there will be size**2 many total nodes.
- ****kwargs** –

  tiling: int Should be provided if the network type is 2D_lattice. Tiling of the 2d lattice. It can be 3,4,6 for now. This is the number of neighbors to be connected.

- ****kwargs** –

periodic: bool Should be provided if the network type is `2D_lattice`. if True, edges of the planar lattice are going to be glued together. See `networkx.grid_2d_graph`.

- ****kwargs** –

  banded: bool Should be provided if the network type is `ring_lattice`. If True, the closest `band_length` many neigbors from right and left is going to be connected to every node, creating a banding.

- ****kwargs** –

  band_length: int Sould be provided if the network type is `ring_lattice`. Geometric degree divided by 2. Note that geometric degree must be an even number.

add_noise_to_geometric(*noise_type*, *d2*)

This method adds non-geometric edges to the network that are long range. Depending on the 'noise_type' the way we add these long range edges differ. If noise_type = `ER_like`, then there will be d2 many non geometric edges ON AVERAGE for every node. When the noise_type = `k_regular`, every node will have exactly d2 many long range edges.

Parameters

- **noise_type** (*str*) – ER_like or k_regular
- **d2** (*int*) – degree to assign non-geometric edges to every node

Returns

Return type None. Updates the geometric_network.A

Raises **ValueError** – if the geometric_network.N * d2 is an odd number.

average_over_trials(*matrix*)

Helper funtion to take the averages over trials of the given matrix.

Parameters **matrix** (*array k x Trials*) – Matrix to take the average over trials. Matrix have to be k x Trials. This can be the size of the contagion or first activation times depending on what you need.

Returns **mean_matrix** – Mean matrix

Return type array k x 1

compute_persistence(*distances*, *spy=False*)

Helper to compute persistent homology using the distance matrix by building a Rips filtration up to dimension 2(topological features to be observed are going to be 1 dimensional at max). First normalizes the distances before the computation.

Parameters

- **distances** (*n x n array*) – distance matrix. First output of the make_distance_matrix.
- **spy** (*bool, optional*) – Take a peak at the persistence diagram

Returns **diag** – The birth and death times of the topological features in all dimensions.

Return type list

display(*n_size=15*, *labels=True*)

Method to pass parameters into `nx.draw()`.

Parameters

- **n_size** (*int*) – node sizes.

- **labels** (*bool*) – node labels.

Returns

Return type nx.draw(self.graph)

display_comm_sizes(*Q*, *labels*)

Helper to visualize the size of the active nodes during the contagion. Shades are indicating the max and min values of the spread starting from different nodes.

Parameters

- **Q** (*list, [n x T+1 array]*) – Output of the make_distance_matrix appended in a list
- **labels** (*figure labels corresponding to every list element, threshold, network type, C etc...*) –

excitation(*T*, *C*, *seed*, *threshold*, *refractory=False*, *ax=None*, *spy=False*)

THE CORE FUNCTION OF THE NEURONAL CONTAGION MODEL.

In this model, a neuron fires if the ratio of it's excited neighbors to the total number of neighbors is greater than the threshold. Let's call the difference between this ratio and the threshold = F so that if F is positive, neuron is going to fire and it doesn't fire when it's negative. We add some stocasticity to the model by defining the sigmoid function so that the probability that the neuron is going to fire is not a step function, but a sigmoid function.

Parameters

- **T** (*int*) – Number of time steps contagions is going to be iterated.
- **C** (*int*) – A positive constant for the sigmoid function, if C is too large(>100), jump from 0 to 1 is gonna be too quick i.e. model is going to be deterministic.
- **seed** (*int*) – node id to start the contagion, in the first time step, we infect the neighbors of the seed with probablity 1 then enter the while loop below
- **threshold** (*float*) – threshold to compare for a neuron's neighbor input. threshold must be in (0,1).
- **refractory** (*bool*) – if TRUE, sets the refractory period of 1 time step i.e. neuron cannot fire for 1 time step right after it fires. if FALSE, neuron stays active once its activated.
- **ax** (*matplotlib.axis, bool*) – if spy is TRUE, there have to be an axis provided to plot the contagion spread.
- **spy** (*matplotlib.axis, bool*) – if spy is TRUE, there have to be an axis provided to plot the contagion spread.

Returns

- **activation_times** (*array*) – An array of n x 1 keeping track of the first time step the corresponding node gets activated.
- **size_of_contagion** (*array*) – An array of (T+1) x 1 keeping track of the number of active nodes at a given time(at t = 0, all neighbors of the seed is active)

make_distance_matrix(*T*, *C*, *threshold*, *Trials*, *refractory*, *spy_distance=False*)

A shortcut to run all of the above functions in one function. This creates an activation matrix by running the contagion on starting from every node and encoding the first activation times of each node. Then, finding the euclidean distances between the columns of this matrix, creating a distance matrix so that the (i,j) entry corresponds to the average time(over the trials) that a contagion reaches node j starting from node i.

Parameters

- **T** (*int*) – Number of time steps contagions is going to be iterated.
- **C** (*int*) – A positive constant for the sigmoid function, if C is too large(>100), jump from 0 to 1 is gonna be too quick i.e. model is going to be deterministic.
- **threshold** (*float*) – threshold to compare for a neuron's neighbor input. threshold must be in (0,1).
- **Trials** (*int*) – Number of trials to run the contagion on the same network.
- **refractory** (*bool*) – if TRUE, sets the refractory period of 1 time step i.e. neuron cannot fire for 1 time step right after it fires. if FALSE, neuron stays active once its activated.
- **spy_distance** (*bool*) – Check True if you want to take a peak at the distance matrix.

Returns

- **D** (*n x n array*) – Distance matrix
- **Q** (*n x T+1 array*) – Array carrying the size of the contagion at every time step.

one_d_Delta(*persistence*)

Helper to compute the specific topological features.

Parameters **persistences** (*list*) – A list of birth and death times of the topological features or the output of the compute_persistence.

Returns

- **Delta_min** (*float*) – The difference between the life times of the longest and the second longest 1-cycle.
- **Delta_max** (*float*) – The difference between the life times of the longest and the shorthes 1-cycle.
- **Delta_avg** (*float*) – The average lifetime of all the 1-cycles.

run_excitation(*Trials*, *T*, *C*, *seed*, *threshold*, *refractory*, *ax=None*)

Helper function to run the excitation over several trials.

Parameters

- **Trials** (*int*) – Number of trials to run the contagion on the same network.
- **T** (*int*) – Number of time steps contagions is going to be iterated.
- **C** (*int*) – A positive constant for the sigmoid function, if C is too large(>100), jump from 0 to 1 is gonna be too quick i.e. model is going to be deterministic.
- **seed** (*int*) – node id to start the contagion, in the first time step.
- **threshold** (*float*) – threshold to compare for a neuron's neighbor input. threshold must be in (0,1).
- **refractory** (*bool*) – if TRUE, sets the refractory period of 1 time step i.e. neuron cannot fire for 1 time step right after it fires. if FALSE, neuron stays active once its activated.
- **ax** (*matplotlib.axis, optional*) – if not provided, contagion will not be shown.

Returns

- **activation_times** (*array*) – An array of n x Trials keeping track of the first time step the corresponding node gets activated for each trial.

- **size_of_contagion** (*array*) – An array of (T+1) x Trials keeping track of the number of active nodes at a given time(at t = 0, all neighbors of the seed is active) for each trial.

spy_first_activation (*first_activation_times*)

Helper function to visualize the first activation times.

Parameters **first_activation_times** (*array of size n x Trials*) – First output of the run_excitation showing the first time step that the contagion reaches to a given node.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

A (*Complex_Contagions.geometric_network attribute*), 13
add_noise_to_geometric() (*Complex_Contagions.geometric_network method*), 14
average_over_trials() (*Complex_Contagions.geometric_network method*), 14

C

compute_persistence() (*Complex_Contagions.geometric_network method*), 14

D

display() (*Complex_Contagions.geometric_network method*), 14
display_comm_sizes() (*Complex_Contagions.geometric_network method*), 15

E

excitation() (*Complex_Contagions.geometric_network method*), 15

G

geometric_network (*class in Complex_Contagions*), 13
graph (*Complex_Contagions.geometric_network attribute*), 13

M

M (*Complex_Contagions.geometric_network attribute*), 13
make_distance_matrix() (*Complex_Contagions.geometric_network method*), 15

N

N (*Complex_Contagions.geometric_network attribute*), 13

O

one_d_Delta() (*Complex_Contagions.geometric_network method*), 16

P

pos (*Complex_Contagions.geometric_network attribute*), 13

R

run_excitation() (*Complex_Contagions.geometric_network method*), 16

S

spy_first_activation() (*Complex_Contagions.geometric_network method*), 17

T

text (*Complex_Contagions.geometric_network attribute*), 13