Neuronal Cascades

Release 0.0.1

Bengier Ülgen Kılıç

Feb 10, 2023

CONTENTS:

`Neuronal Cascades` is a python package for simulating spreading processes, such as *Watts-Thresholds model* [1,2] or *Simplicial Threshold model* [3] on networks. For example, for STM cascades, a vertex v_i becomes active only when the activity across its simplicial neighbors surpasses a threshold T_i. See the paper [3] for details.

ref [1] - Watts, Duncan J. A simple model of global cascades on random networks, PNAS, 99, 9, 2002, 10.1073/pnas.082090499.

ref [2] - Taylor, D., Klimm, F., Harrington, H. et al. Topological data analysis of contagion maps for examining spreading processes on networks. Nature Communications, 6, 7723 (2015). https://doi.org/10.1038/ncomms8723

ref [3] - Kilic, B.Ü., Taylor, D. Simplicial cascades are orchestrated by the multidimensional geometry of neuronal complexes. Communications Physics, 5, 278 (2022). https://doi.org/10.1038/s42005-022-01062-3

CHAPTER

ONE

INTRODUCTION

Here, we develop computational framework for interplay between the dynamics (spreading process) and network topology manifesting through higher-order connections embedded in a manifold structure for neuronal activity. One can study spatio-temporal patterns of STM cascades over noisy geometric complexes, which contain both short- and long-range simplices and are a generalization of noisy geometric networks.

In this module, one can investigate the dynamics of a Simplicial Threshold model (STM) starting from a seed cluster and spreading across the underlying simplicial complex. 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 cascade models. There are 3 main parameter groups summarized under *Network paramaters*, *Dynamics parameters* and *Neuron parameters*.

1.1 Geometric and Noisy Geometric Ring complexes

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

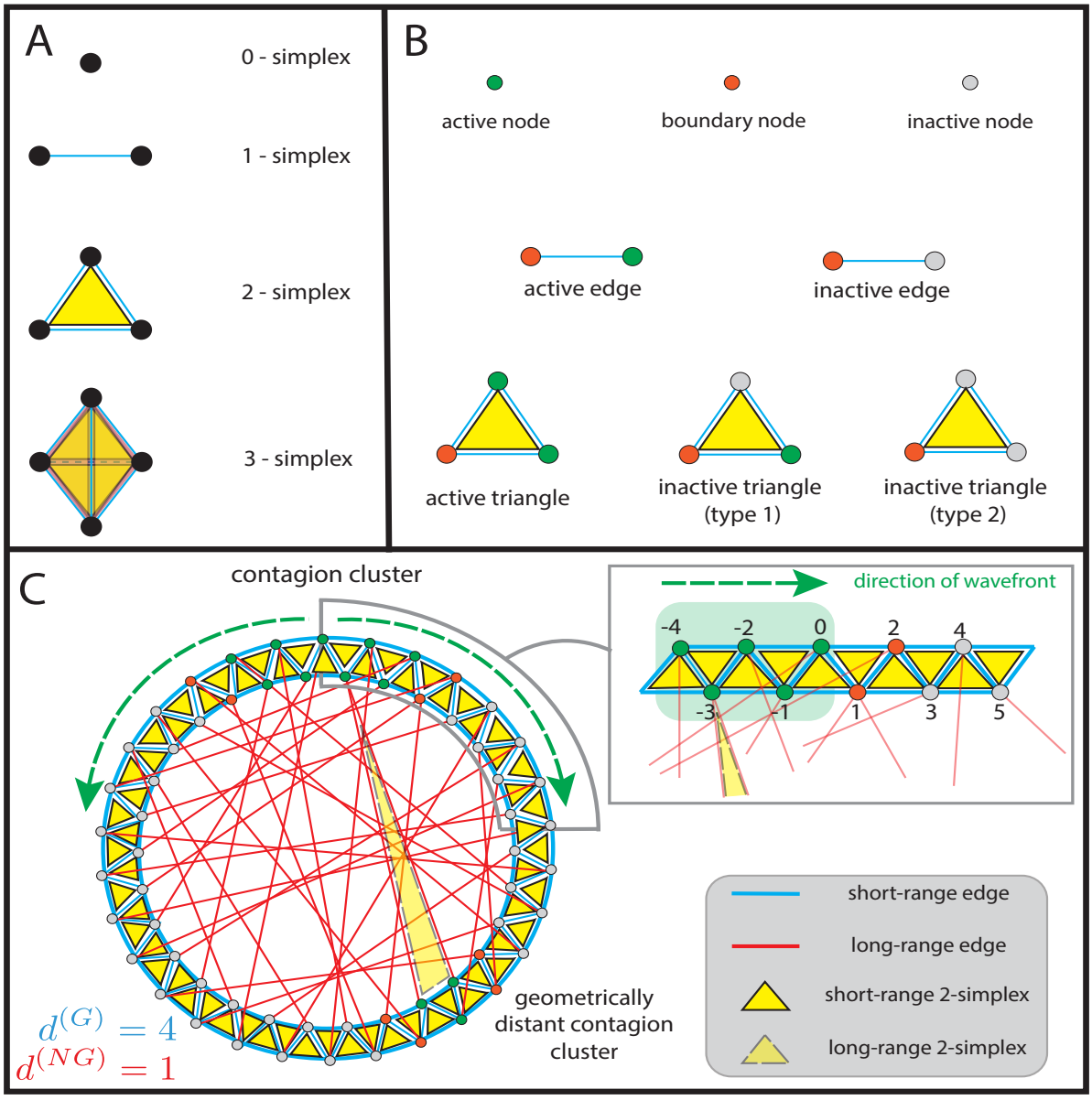
Noisy geometric networks are obtained by connecting 'distant' vertices of the geometric network. These network topology manipulations are shown to demonstrate various contagion spread phenomenans such as wavefront propagation (WFP) or appearance of new clusters (ANC) in these networks.

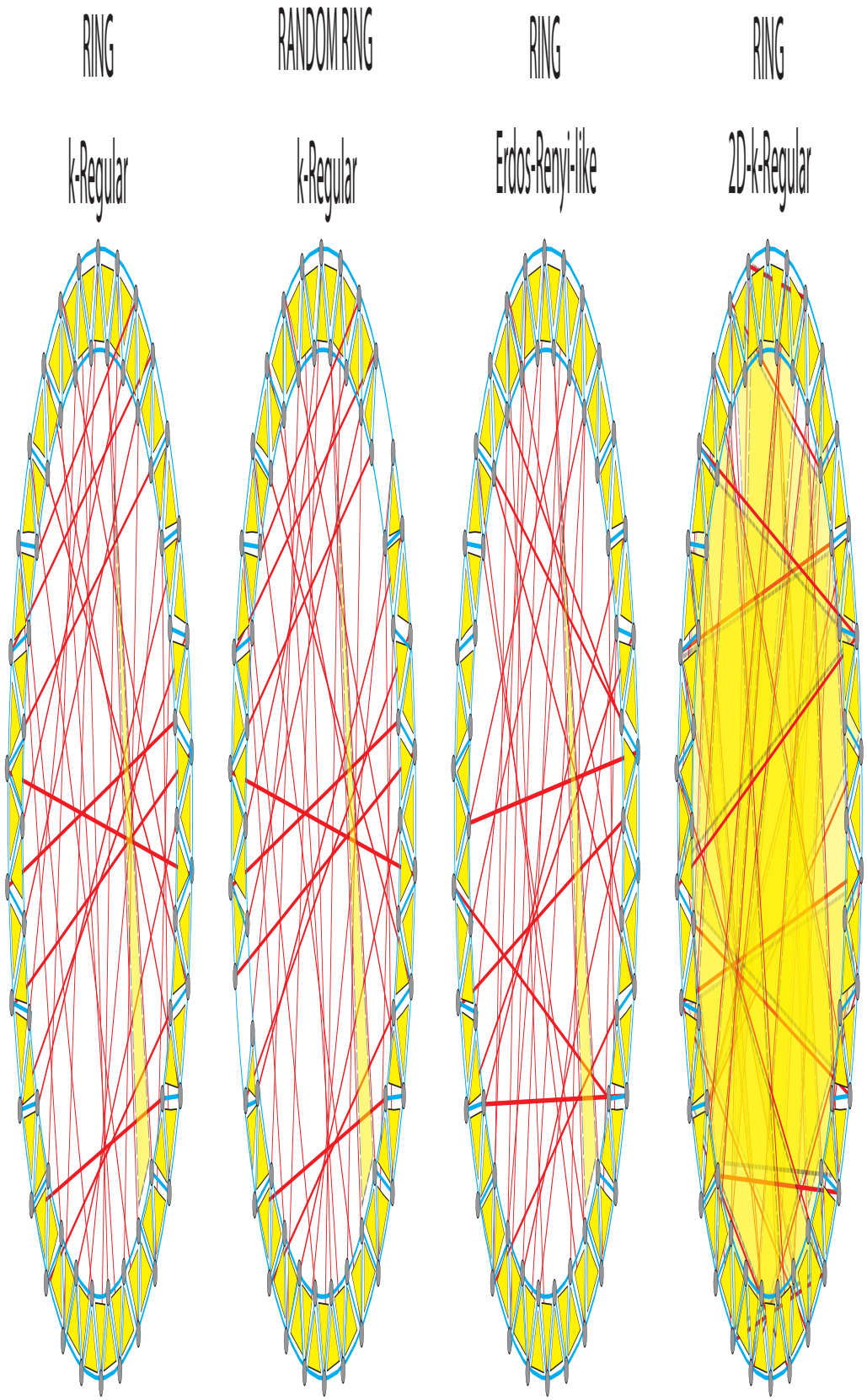
A noisy ring complex involves vertices that lie along a 1D manifold that is embedded in a 2D ambient space. (Vertices are placed slightly alongside the manifold to allow easy visualization of 2-simplices.) Each vertex has $d^{(G)}$ geometric edges to nearby vertices and $d^{(NG)}$ nongeometric edge to a distant vertex. Higher-dimensional simplices arise in the associated clique complex and are similarly classified. An STM cascade exhibits WFP when it progresses along the ring manifold, and ANC events when it jumps across a long-range edge or higher-dimensional simplex.

```
#### NETWORK PARAMETERS ####
size = 400 # number of neurons
GD = 10 # geometric degree
nGD = 4 # non-geometric degree
topology = 'Ring' or 'random_Ring'
noise_type = 'k-regular' or 'ER-like' or '2D_k-regular'
```

Alternatively, one can input any adjacency matrix in order to run simulations on other networks that are not generated by the above options. In that case, network parameters can be modified as below.

```
#### NETWORK PARAMETERS ####
size = 400
G = nx.grid_2d_graph(size, size)
topology = 'lattice'
matrix = nx.adjacency_matrix(G).todense()
```


A
0 - simplex
1 - simplex
2 - simplex
3 - simplex
B
active node
boundary node
inactive node
active edge
inactive edge
active triangle
inactive triangle (type 1)
inactive triangle (type 2)
C
contagion cluster
direction of wavefront
-4
-2
0
2
4
-3
-1
1
3
5
short-range edge
long-range edge
short-range 2-simplex
long-range 2-simplex
geometrically distant contagion cluster
$d^{(G)} = 4$
$d^{(NG)} = 1$


RING
k-Regular
RANDOM RING
k-Regular
RING
Erdos-Renyi-like
RING
2D-k-Regular

1.2 Simplicial Threshold Model

We are inspired by neuoronal cascades to asses the spreading phenomena. The core function that we run our experiments decides if a given neuron is going to fire or not by a sigmoid function $f(R_i, C) = \frac{1}{1+\exp^{-C.R_i}}$ where R_i, the simplicial exposure, is a function of current network history defined by $R_i = \left[(1-K) * \sum_{e \in E_i} \frac{e}{d_i^e} + (K) * \sum_{t \in T_i} \frac{t}{d_i^t}\right] - \tau_i$ where E_i is the set of active edge neighbors, T_i is the set of active triangle neighbors of node i, d_i^e and d_i^t are edge and triangle degrees of node i respectively. The constant K ,2-simplex influence, is used to strike a balance between traditional activation maps and higher order, or simplicial, cascade maps.

The main class we use `Geometric_Brain_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. Moreover, $K = 0$ recovers an edge contagion whereas $K = 1$ recovers a pure triangle contagion.

```
#### DYNAMICS PARAMETERS ####
K = 0.5 # 2-simplex influence ranging between 0 and 1. edge-dominant model if 0,␣
↪triangle-dominant model if 1.
C = 10000 # Stochasticity parameter. Higher the more deterministic
TIME = 500 # Number of discrete time-steps to run one single cascade
seed = 200 # seed node to initialize the cascade
```

1.3 Neuronal Subtypes

In the package, `Geometric_Brain_Network` object has a subclass called `neuron` which can have individual activation thresholds as well as memory and refractory periods as a function of discrete time steps. This generalization enables heterogenity in the experiments as well as complexity of the non-trivial interactions.

```
#### NEURON PARAMETERS ####
threshold = 0.1 # vertex activation threshold
memory = TIME # number of discrete time steps that neuron stays active once they are␣
↪active. If 0, neuron will stay active only 1 time step.
rest = 500 # number of discrete time-steps that neuron stays in the refrectaroy period.␣
↪In this state, neuorons are not allowed to get active.
```

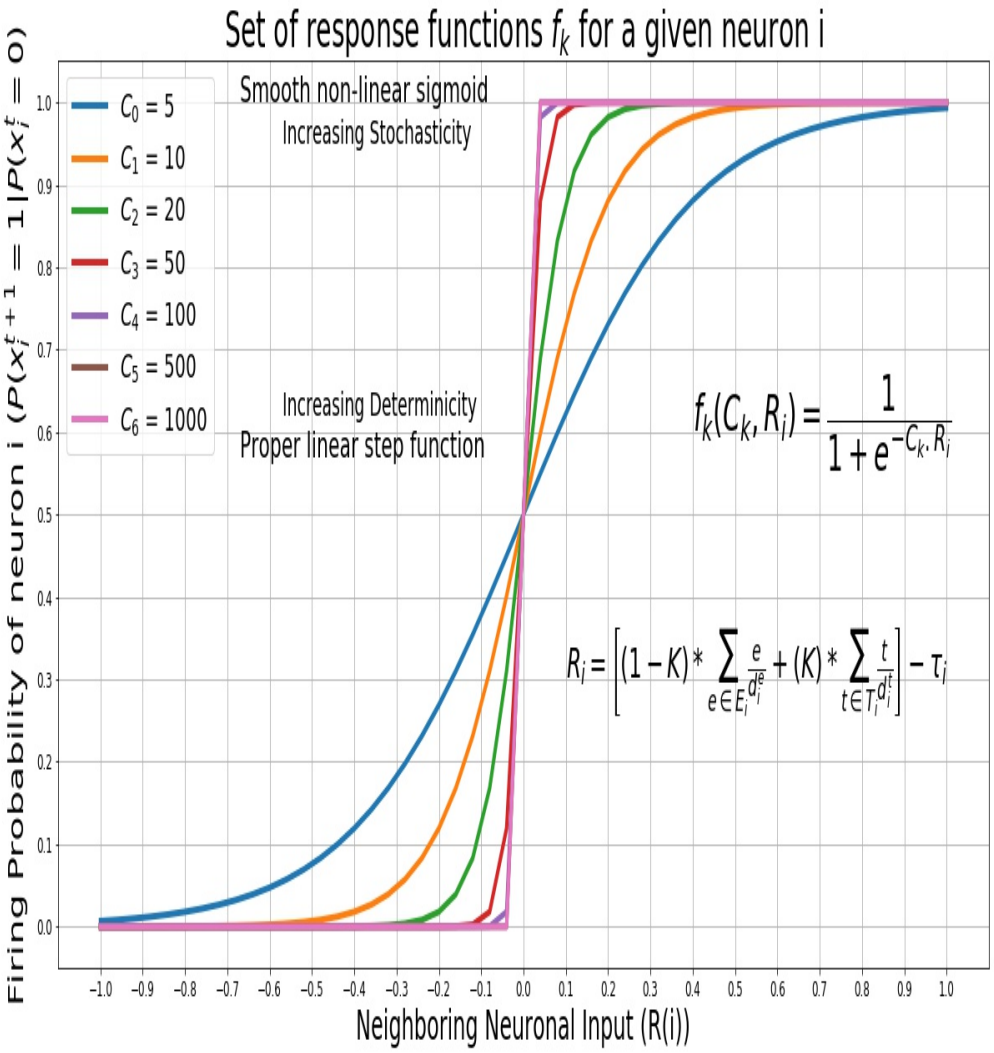
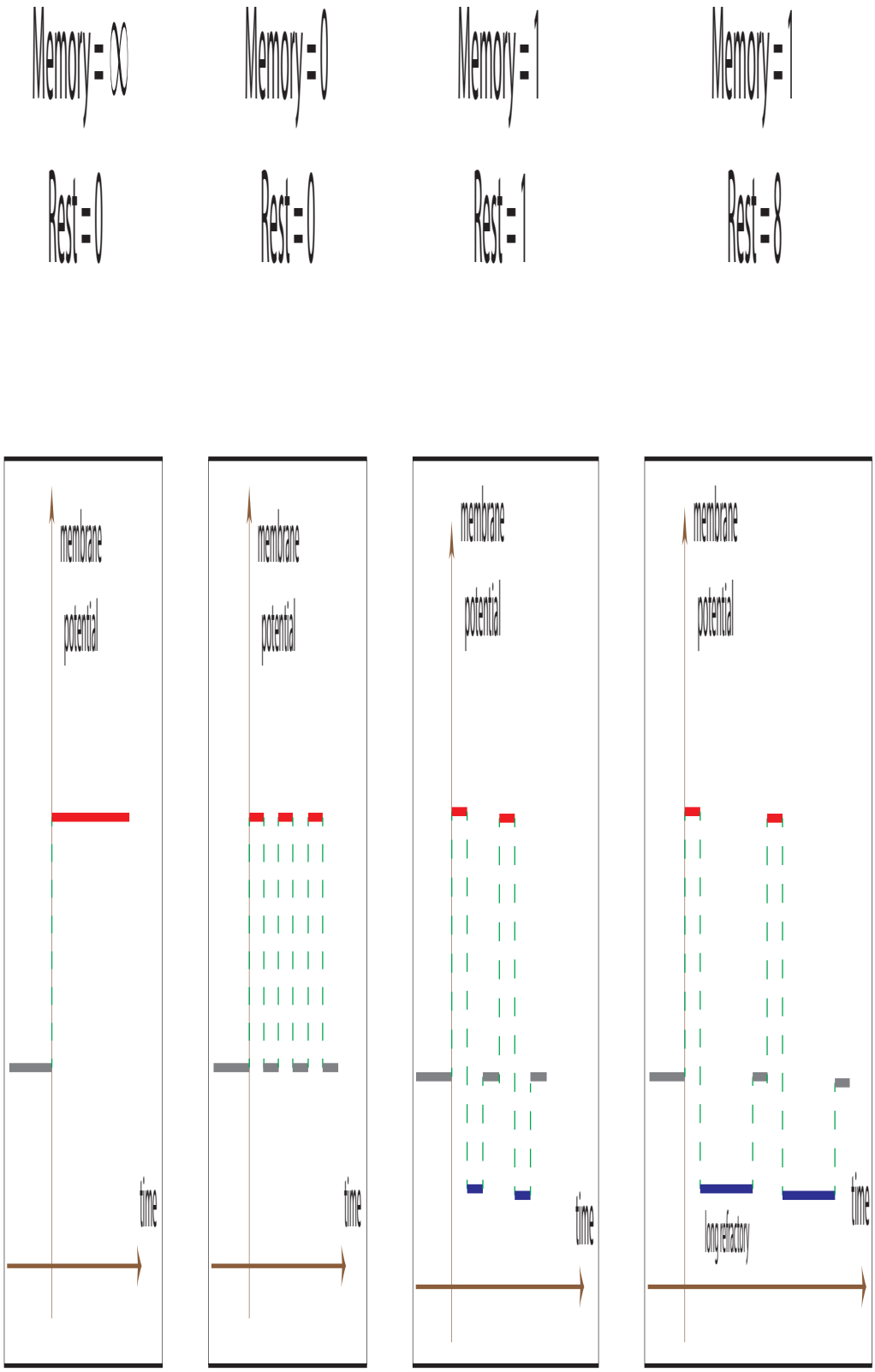


Fig. 1: Set of neuronal activation functions as a function of C.


Memory = ∞
Rest = 0
Memory = 0
Rest = 0
Memory = 1
Rest = 1
Memory = 1
Rest = 8
membrane potential
time
membrane potential
time
membrane potential
time
membrane potential
time
long refractory

CHAPTER

TWO

TUTORIAL

2.1 Initiate a Geometric_Brain_Network object

Create a simplicial ring complex on a ring. Topology is only available for a ring now. GD, geometric degree, is the local neighbors of a neurons whereas nGD, nongeometric degree, is the distant neighbors of a neuron.

```
#### NETWORK VARIABLES
size = 400
GD = 10
nGD = 4
topology = 'Ring'
noise = 'k-Regular'
network = Geometric_Brain_Network.Geometric_Brain_Network(size, geometric_degree = GD,␣
↪nongeometric_degree = nGD, manifold = topology, noise_type = noise)
```

2.2 Inheriting neuron objects

Define neuronal properties and then use `get_neurons` to inherit individual neurons into the network.

```
    #### EXPERIMENT VARIABLES
TIME = 100 ## number of iterations
seed = int(size/2) ## seed node
C = 10000 ## constant for tuning stochasticity(high C yields deterministic experiments)
K = 0 ## constant weighing the edges vs triangles K=0 pure edge contagions, K=1 pure␣
↪triangle contagion

#NEURON VARIABLES
threshold = 0.2 # node threshold
memory = TIME ##When a node is activated, it stays active forever(SI model) when memory␣
↪= TIME.
rest = 0# neurons don't rest

##INITIATE NEURONS and Inherit them
neurons = [Geometric_Brain_Network.neuron(i, memory = memory, rest = rest, threshold =␣
↪threshold) for i in range(size)]
network.get_neurons(neurons)## this is for runnning experiments with new set of neurons␣
↪without changing the network
```

2.3 Run a single example cascade

Core function `run_dynamic` runs an experiment with given variables.

```
activation1, Q1 = BN.run_dynamic(seed, TIME, C, K)
```

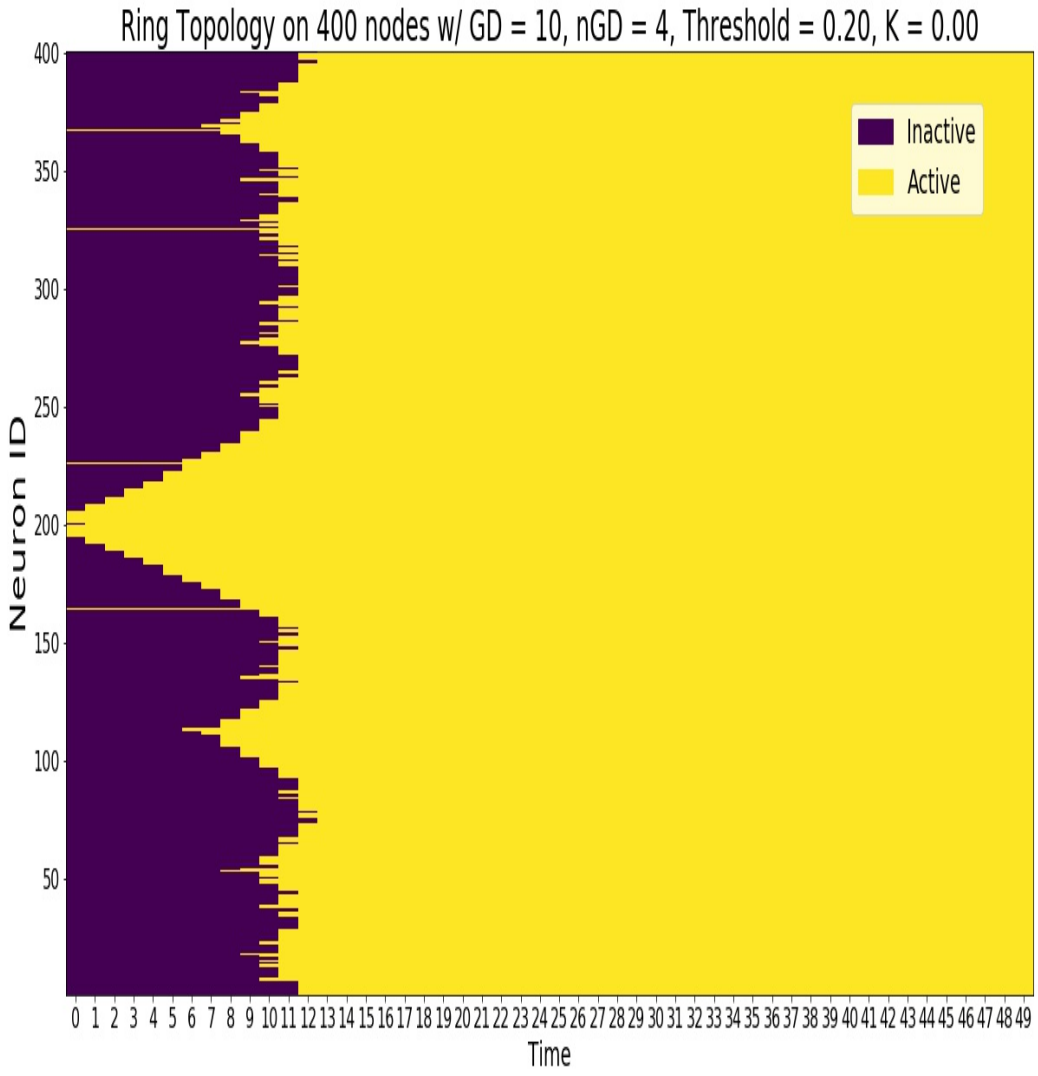


Fig. 1: A single experiment starting at the seed node 200. Initial wavefront propagation can be observed.

2.4 Running experiments without changing the network connectivity

One may want to work with a different set of experiment or neuronal variables without changing the underlying topology. This is when `get_neurons` function comes handy.

```
## with a new set of variables you can run a new experiment without changing the network
K = 0
threshold = 0.3
memory = TIME
rest = 0

neurons_2 = [neuron(i, memory = memory, rest = rest, threshold = threshold) for i in␣
↪range(size)]
BN.get_neurons(neurons_2)

activation2, Q2 = BN.run_dynamic(seed, TIME, C, K)
```

2.5 Running simplicial cascades

Simplicial cascades can be ran by simply varying the parameter K between 0 and 1.

```
## with a new set of variables you can run a new experiment without changing the network
K = 1
threshold = 0.2
memory = TIME
rest = 0

neurons_3 = [neuron(i, memory = memory, rest = rest, threshold = threshold) for i in␣
↪range(size)]
BN.get_neurons(neurons_3)

activation3, Q3 = BN.run_dynamic(seed, TIME, C, K)
```

2.6 Neurons with memory and refractory period

Our model is as general as it can be. So, neurons can have arbitrary number of memory or refractory period given in discrete time steps. This generalization increases complexity of the dynamics really quick.

```
K = 0.5 # average of edge and triangle contagions
memory = 1## memory of a neuron is how many time steps neurons are going to stay active␣
↪after they activated once
rest = 0#rest of a neuron is how many time steps neurons are going to be silent after␣
↪they run out of memory, refractory period.
threshold = 0.2

neurons_4 = [neuron(i, memory = memory, rest = rest, threshold = threshold) for i in␣
↪range(size)]

BN.get_neurons(neurons_4)
```

(continues on next page)

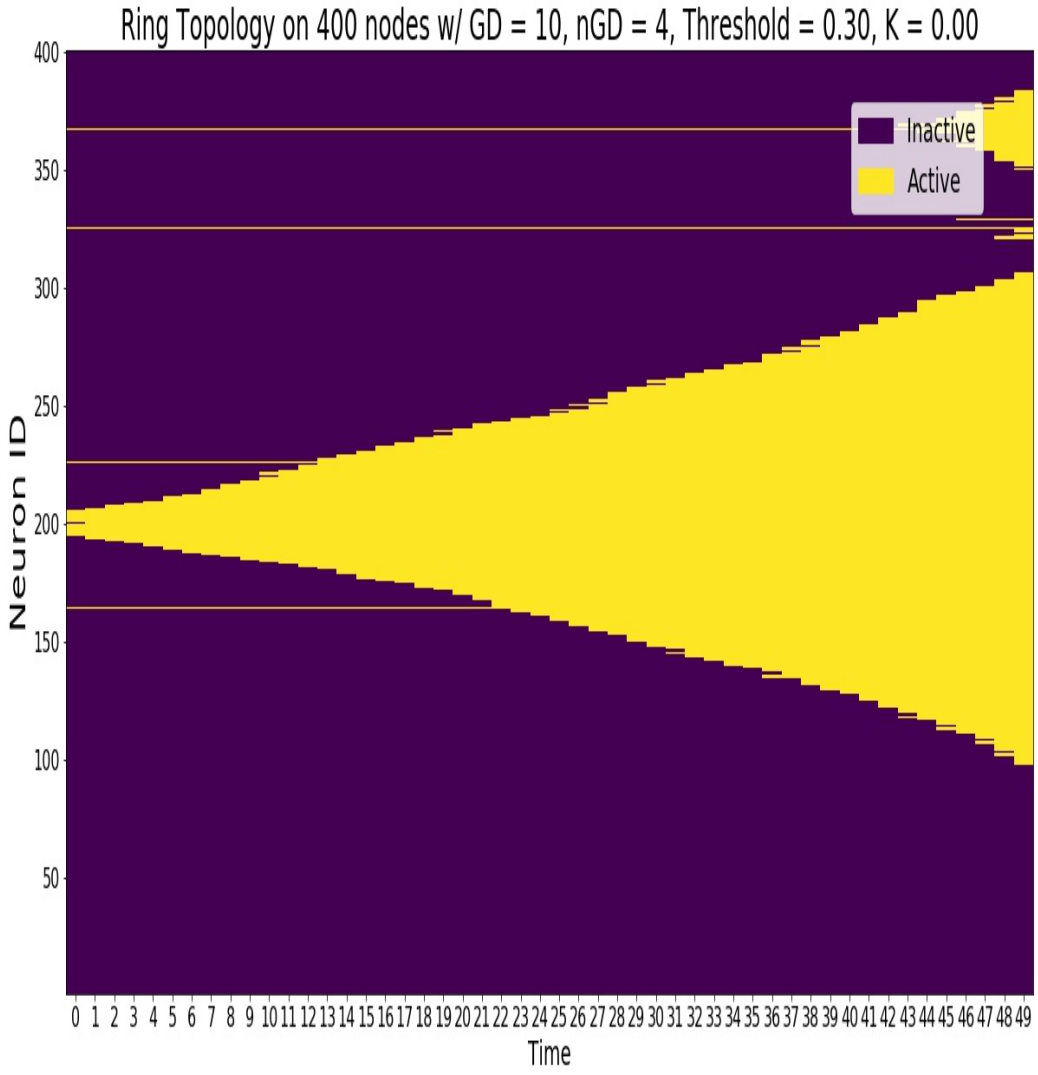


Fig. 2: We increased the global node thresholds to 0.3 which slowed down the signal, wavefront.

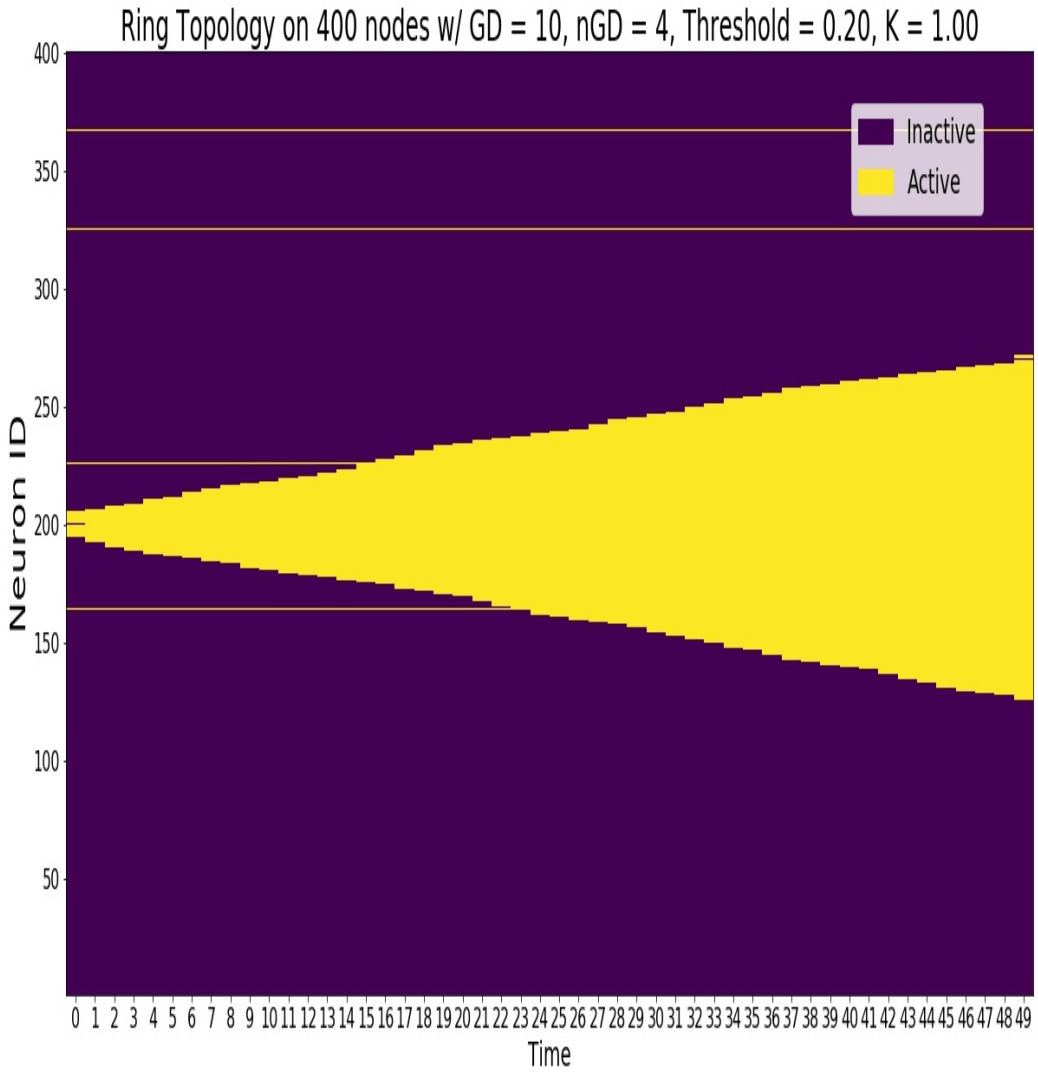


Fig. 3: Even though the global node threshold is 0.2 we observe a slow signal. The reason is that we set K=1 which implies a full triangle contagion.

(continued from previous page)

```
activation4, Q4 = BN.run_dynamic(seed, TIME, C, K)
```

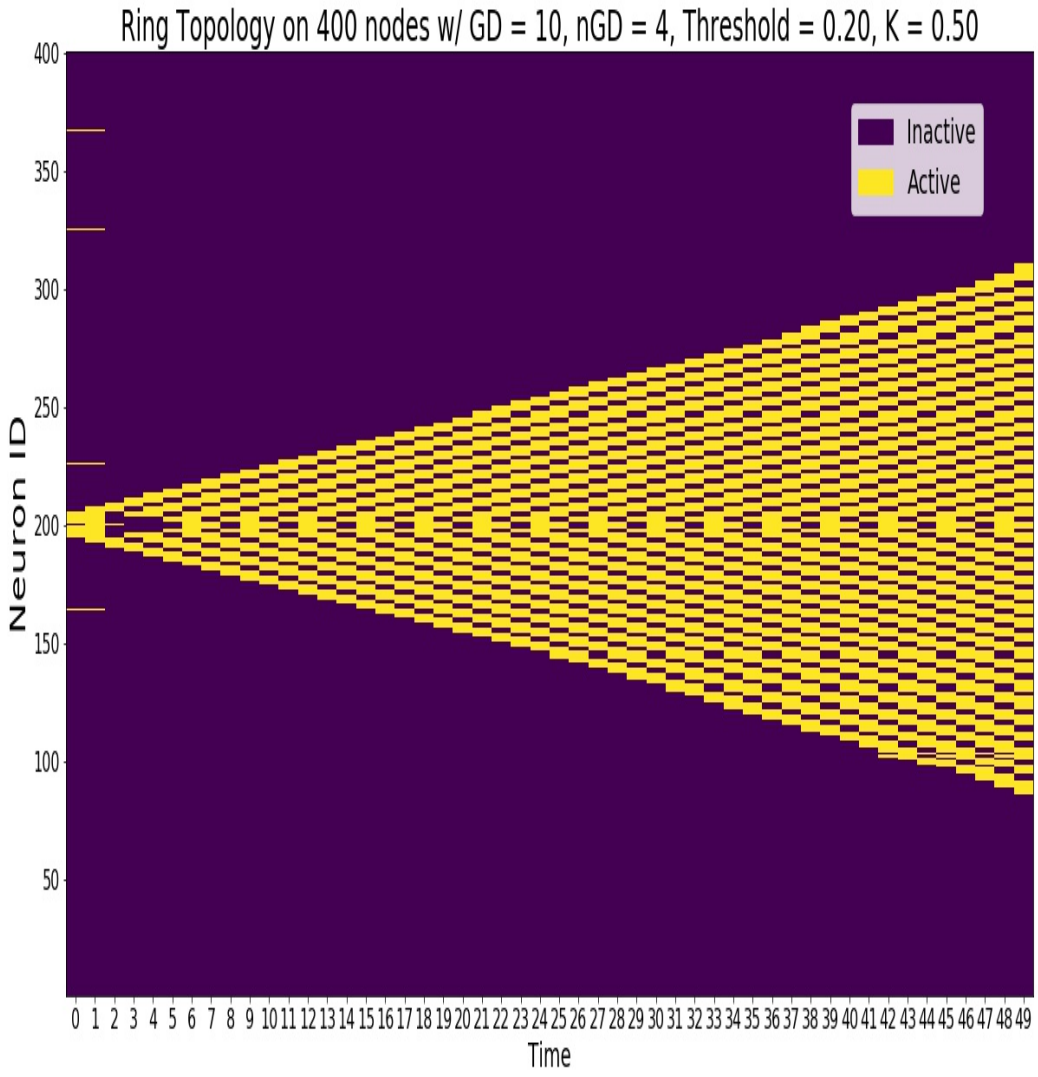


Fig. 4: Slow signal propagation where neurons are active only 1 time step. Signal spreads as the neurons blink.

2.7 Running stochastic models

Stochasticity of the neuronal responses can be adjusted using the experiment variable C. Higher values make the system deterministic.

```
K = 1 ## triangle contagion
memory = 2## memory of a neuron is how many time steps neurons are going to stay active␣
↪after they activated once
rest = 1#rest of a neuron is how many time steps neurons are going to be silent after␣
↪they run out of memory, refractory period.
threshold = 0.2
C = 10 ## make the system stochastic, higher values(C>500) is going to make the system␣
↪deterministic

neurons_5 = [neuron(i, memory = memory, rest = rest, threshold = threshold) for i in␣
↪range(size)]

BN.get_neurons(neurons_5)

activation5, Q5 = BN.run_dynamic(seed, TIME, C, K)
```

2.8 Looking at the cascade size

We can plot the size of the active nodes as a function of time.

```
Q = [Q1,Q2,Q3,Q4,Q5]
fig, ax = BN.display_comm_sizes_individual(Q,labels)
```

2.9 Run a full scale experiment

In order to asses global features, we run experiments for every seed node i and obtain the activation times for every neuron j i.e. create a distance matrix whose (i,j) entry is the first time the node j is activated on a contagion starting from i. Distance matrices enable a global scale TDA analysis.

```
FAT, CS = BN.make_distance_matrix(TIME, C, K)
```

2.10 Persistence diagrams

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

```
delta_min, delta_max = BN.compute_persistence(FAT, spy = True)##returns the lifetime␣
↪difference of the longest living one cycles(delta_min) and lifetime difference of the␣
↪longest and shorthest living one cycles(delta_max)
```

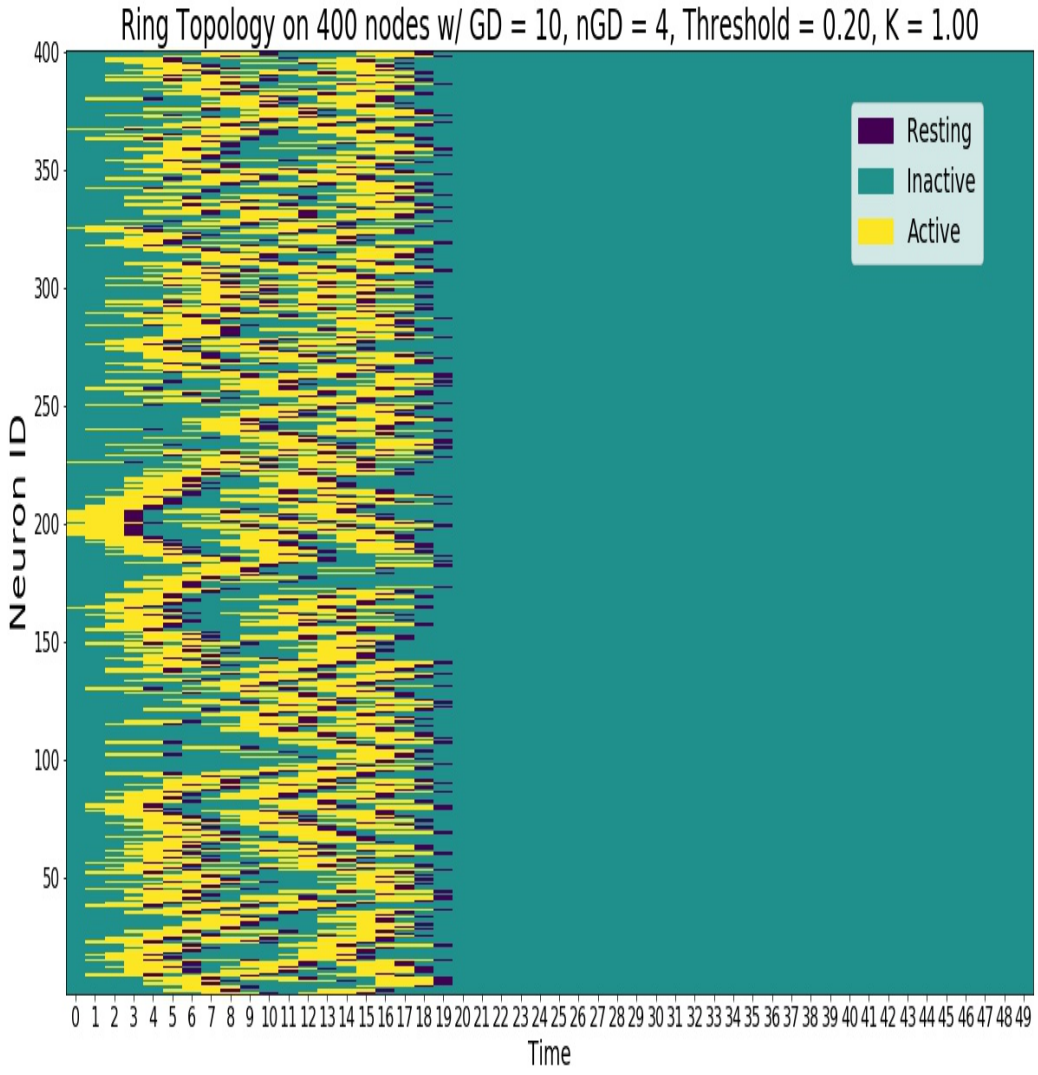


Fig. 5: As the refractory period is nonzero, complexity of the system increases exponentially.

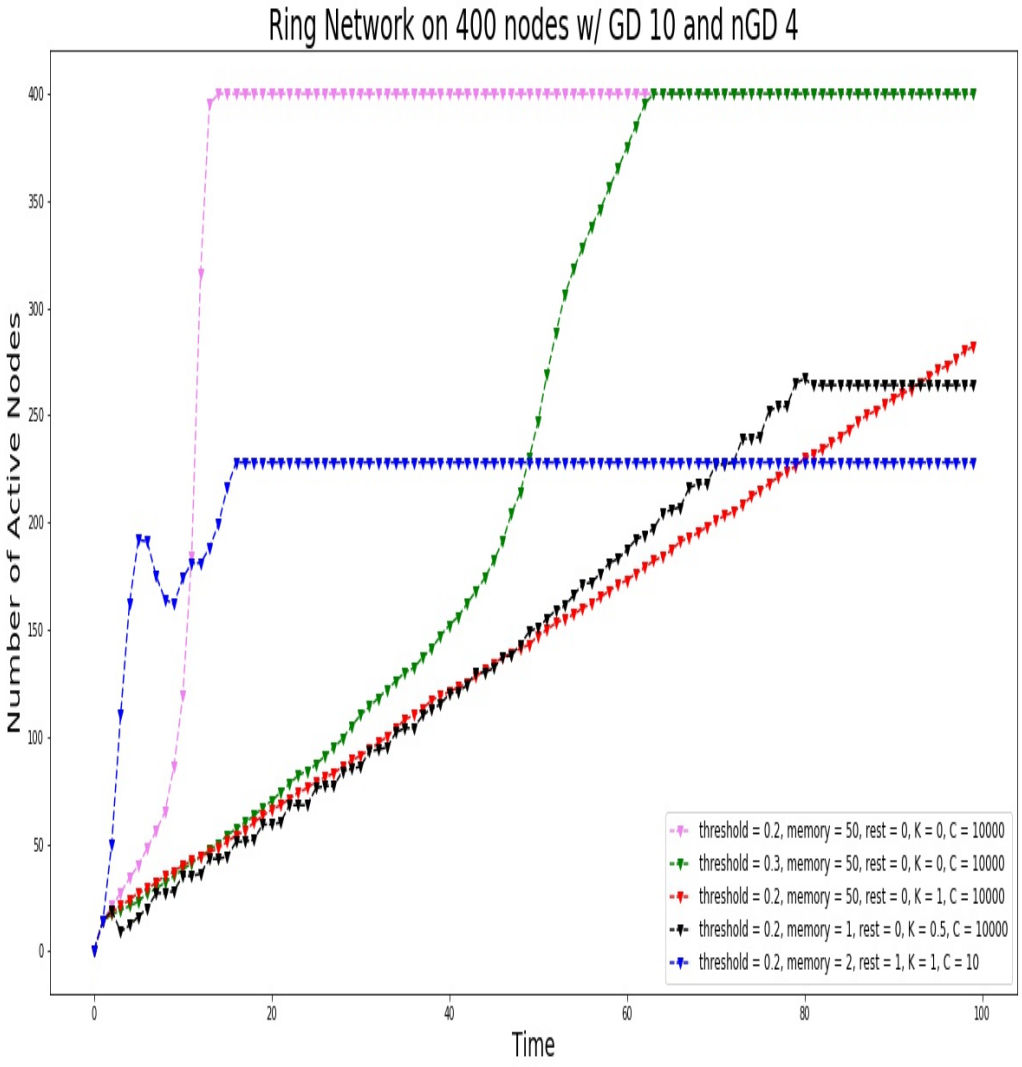


Fig. 6: Spread of the signal as a function of active neurons.

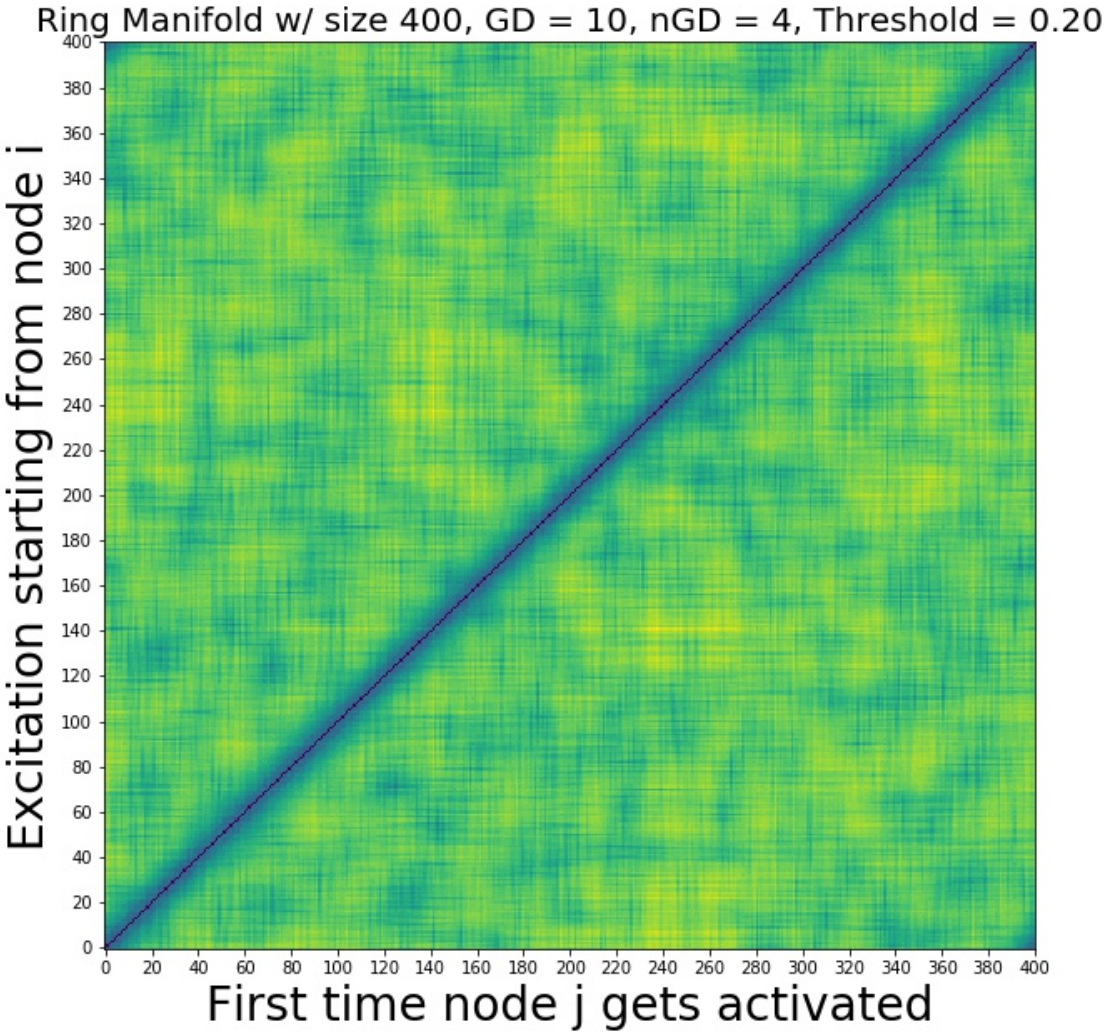


Fig. 7: The distance matrix. The input for the persistent homology.

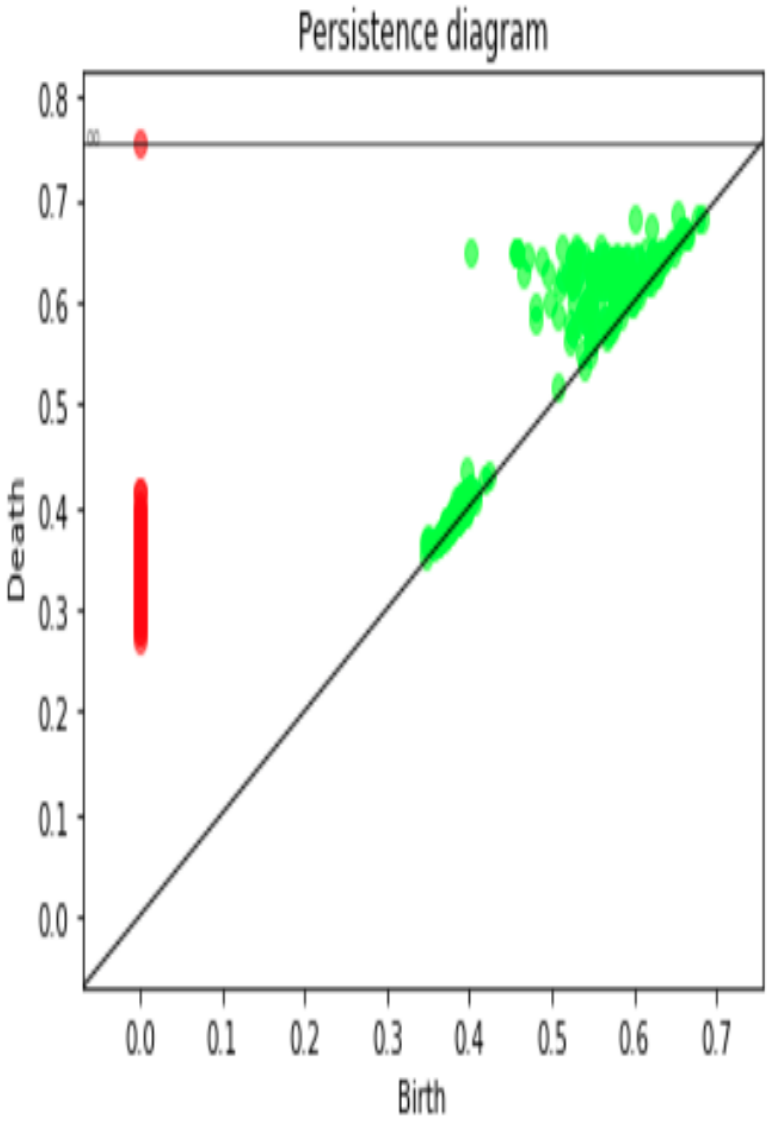


Fig. 8: Persistence diagram computed from the distance matrix via Rips filtration. Green is 1-D features, red is 0-D features.

CHAPTER

THREE

SEMANTICS OF NEURONAL CASCADES

class Geometric_Brain_Network.**Geometric_Brain_Network**(*size*, *geometric_degree=1*, *nongeometric_degree=0*, *manifold='Ring'*, *noise_type='k-regular'*, *matrix=None*, *perturb=0*, *higher_order=False*)

Bases: `object`

Geometric Brain Network object to run simplicial cascades on.

N
Size, number of nodes in the network.

Type int

GD
Geometric degree of the network.

Type int

nGD
non-Geometric degree of the network.

Type int

manifold
The geometric topology of the network. Only 'Ring' is available currently.

Type str

text
Summary of the network.

Type str

A
Adjacency matrix of the graph.

Type array `n x n`

nodes
A list of `neuron` objects that corresponds to the nodes of the network in which IDs of the neurons match with the IDs of the nodes.

Type List

time
An intrinsic time property to keep track of number of iterations of the experiments.

Type int

triangles
A dictionay of the triangles of the network where keys are node ids and values are lists of pairs of node ids that makes up a triangle together with the key value.

Type dict

higher_order
Flag if a higher-order experiment is to be run, that is K>0.

Type Boolean

Parameters

- **size** (*int*) – Size of the network to be initialized.
- **geometric_degree** (*int*) – Uniform number of local neighbors that every node has.
- **nongeometric_degree** (*int*) – Fixed number of distant neighbors that every node has.
- **manifold** (*str*) – Type of the network to be created. If 'Ring' or 'random_Ring' then a syntehtic ring network will be created, if 'lattice' then *matrix* argument can be used to input any adjacency matrix.
- **noise_type** (*str*) – k-regular or er-like
- **matrix** (*array-like*) – Argument for inputting and adjacency matrix, ff *manifold* is 'lattice'.
- **perturb** (*int*) – Number of edges per vertex that the local manifold is to be perturbed.
- **higher_order** (*Boolean*) – Flag for higher-order experiments. Since extracting the triangles is costly, when running an edge-based model, we don't have to compute them.

ablate_geo_triangles(*A*)
Helper to remove links from the geometric strata. *self.perturb* many links per vertex will be removed. :param A: Adjacency matrix of the network. :type A: array

Returns **A** – Perturbed adjacency matrix.

Return type array

add_noise_to_geometric()
This method adds non-geometric edges to the network that are long range. Every node will have nGD many nongeometric, long range, edges. Options are 'k-regular', 'ER_like' and '2D_k-regular'.

compute_persistence(*distances*, *dimension*, *spy*)
Helper to compute persistent homology using the distance matrix by building a Rips filtration up to given dimension(topological features to be observed are going to be one less 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.
- **dimension** (*int*) – Max dimension of the topological features to be computed.
- **spy** (*bool, optional*) – Take a peak at the persistence diagram.

Returns

- **Delta_min** (*array*) – Difference of the lifetimes between longest and second longest living two 1-cycles.

- **Delta_max** (*array*) – Difference of the lifetimes between longest and shortest living two 1-cycles.

`display_comm_sizes`(*Q*, *labels*, *TIME*, *C*, *threshold*, *K*)

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, seed node variations.

Parameters

- `Q` (*list, [n x T+1 array]*) – Output of the make_distance_matrix appended in a list
- `labels` (*list*) – Figure labels corresponding to every list element for different thresholds.
- `TIME` (*int*) – A limit on the number of iterations.
- `C` (*int*) – Constant for tuning stochasticity. Higher values yield a deterministic model whereas lower values yield a stochastic model.
- `K` (*float*) – Constant for weighing the edge and triangle activations.

Returns

- **fig** (*matplotlib object*) – Figure to be drawn.
- **ax** (*matplotlib object*) – Axis object for the plots.

`get_neurons`(*neurons*)

Sometimes we want to run experiments on a fixed network without changing the network connectivity. In this case, we can initialize a new set of neurons and use this method to inherit them in the network– changing only the neuronal properties but not the connectivity. :param neurons: A list of `neuron` objects. :type neurons: list

Raises `ValueError` – If the number of neurons and the size of the network doesn't match.

`get_nodes_unique_triangles`(*nonunique_triangle_list*, *i*)

Helper function for finding triangles that flags if a triangle is repeated.

Parameters

- `nonunique_triangle_list` (*array*) – Output of *get_nonunique_triangle_list*.
- `i` (*int*) – Index of the triangle whose repeated triangle neighbors to be removed.

Returns

- **nonunique_triangle_list[tri_flag,1** (*] : array*) – Removed triangles for a given node.
- **tri_flag** (*int*) – flag

`get_nonunique_triangle_list`(*A*)

Helper method finding all of the triangles in the network including the repeated ones.

Parameters `A` (*array*) – Adjacency matrix of the network

Returns **triangle_list** – All triangles in the network.

Return type array

`initial_spread`(*seed*)

Helper method to activate the neighbors of the seed node with probablity 1.

Parameters `seed` (*int*) – Node ID of the seed node.

`make_distance_matrix`(*TIME*, *C*, *K*)

Main function if you are running experiments for a full set of seed nodes. 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

- **TIME** (*int*) – A limit on the number of iterations.
- **C** (*int*) – Constant for tuning stochasticity. Higher values yield a deterministic model whereas lower values yield a stochastic model.
- **K** (*float*) – Constant for weighing the edge and triangle activations.

Returns

- **distance_matrix** (*array*) – `n x n` array with entries the activation times of contagions starting from node i reaching to node j.
- **Q** (*array*) – `n x t` array with entries number of active nodes at every time step for contagions starting at different seeds.

`make_geometric()`
Method for creating a geometric ring network. Options are either 'Ring' or 'random_Ring'. This will be called upon initialization automatically.

`neighbor_input(node_id, K)`
This is a key function as it computes the current input from neighbors of a given node, v_{i}.

Parameters

- **node_id** (*int*) – ID of the node whose input is going to be calculated.
- **K** (*float*) – Constant for weighing the edge and triangle activations.

Returns **F** – Neighboring neuronal input.

Return type float

`refresh()`
Helper method for setting the network time and tolerance to 0. This is necessary between different experiments for any set of parameters including `seed`. Also, calls `refresh_history` which clears `neuoron` histories.

Returns **tolerance** – Tolerance for experiments getting stuck at some point during contagion. Set to 0 at every trial.

Return type int

`return_triangles()`
Function for getting the triangles in the network. This will be automatically called upon initialization of `Geometric_Brain_Network`.

Returns **triangles** – A dictionay of the triangles of the network where keys are node ids and values are lists of pairs of node ids that makes up a triangle together with the key value.

Return type dict

`run_dynamic(seed, TIME, C, K)`
Core function that runs the experiments. There are couple control flags for computational efficiency. If `self.time` exceeds `TIME`, flag. If there is no active neurons left in the network, flag. If everything gets activated once, flag. If `tolerance` exceeds 10, flag i.e. network repeats the exact state of itself 10 times.

Parameters

- **seed** (*int*) – Node ID of the seed node.
- **TIME** (*int*) – A limit on the number of iterations.

- **C** (*int*) – Constant for tuning stochasticity. Higher values yield a deterministic model whereas lower values yield a stochastic model.
- **K** (*float*) – Constant for weighing the edge and triangle activations.

Returns

- **activation_times** (*array*) – Activation times of all the nodes for contagions starting from seed.
- **size_of_contagion** (*array*) – Number of active nodes at every iteration.
- *number_of_clusters* – Number of distinct cascade clusters.

sigmoid(*node_id*, *C*, *K*)

Sigmoid function which adjusts the stochasticity of the neurons depending on *C*.

Parameters

- **node_id** (*int*) – ID of the node whose input is going to be calculated.
- **C** (*int*) – Constant for tuning stochasticity. Higher values yield a deterministic model whereas lower values yield a stochastic model.
- **K** (*float*) – Constant for weighing the edge and triangle activations.

Returns **Z** – Probability of firing.

Return type float

stack_histories(*TIME*)

Helper function for equalizing, stacking, the lengths of histories of ``neuron``s. Comes handy for visualizing single experiments.

Parameters **TIME** (*int*) – Number of discrete time steps for neuron histories to be visualized

Returns **all_history** – `N x TIME` matrix encoding histories of neurons.

Return type array

update_history(*node_id*, *C*, *K*)

Helper method to update the history of the `neuron` objects at every iteration.

Parameters

- **node_id** (*int*) – ID of the node whose history is going to be updates.
- **C** (*int*) – Constant for tuning stochasticity. Higher values yield a deterministic model whereas lower values yield a stochastic model.
- **K** (*float*) – Constant for weighing the edge and triangle activations.

update_states()

Helper method to update the states of `neuron` objects at every iteration.

Returns

- **excited** (*list*) – List of active neurons at the current `time`.
- **ready_to_fire** (*list*) – List of neurons that are in the inactive state and ready to fire at `time+1`.
- **rest** (*list*) – List of neurons that doesn't belong to either of those categories. This is empty as long as there are no refractory period.

class Geometric_Brain_Network.**neuron**(*name*, *state*, *memory*, *rest*, *thresold*)

Bases: *Geometric_Brain_Network.Geometric_Brain_Network*

Neuron objects corresponding to the nodes of `Geometric_Brain_Network`. This is a subclass of `Geometric_Brain_Network`.

name

Neuron ID.

Type int

state

State of a neuron, can be 0,1 (or -1 if `rest` is nonzero).

Type int

memory

Memeory of a neuron. Once a neuron is activated, it is going to stay active `memory` many more discrete time steps(so `memory + 1` in total).

Type int

rest

Refractory peiod of a neuron in terms of discrete time steps.

Type int

threshold

Threshold of a neuron, resistance to excitibility.

Type int

history

History of a neuron encoding the states that it has gone through.

Type list

Parameters

- **name** (*str*) – Neuron ID
- **state** (*int*) – State of a neuron(1 Active, 0 Inactive and -1 rest, refractory).
- **memory** (*int*) – Number of discrete time steps a neuron is going to stay active once it is activated.
- **rest** (*int*) – Refractory period of a neuron in discrete time steps.
- **threshold** (*float*) – Threshold of a neuron.

refresh_history()

Helper method that sets the history of every neuron to an empty list. It is called after `refresh`.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

S

T

U