**Max Bluhm Spring 2024 Documentation and Write-Up**

The project I worked on this spring was to use and adapt the Leaky-Integrate and Fire Model made by Isaac Mineo and Prof. Jennifer Crondelle to generate simulated datasets of the spike firing patterns/rates of ~50 neurons (although the number of neurons can easily be changed) as the neurons are stimulated by a stimulation vector produced by an inhomogeneous poisson process and modulated by a sine wave current representing infraslow oscilations.

This output was then given to the ensemble detector made by Russo and Dorwitz., The intention being to use the simulated data to test the ensemble detector so that we can feel confident about the results generated by the ensemble detector when giving it *in-vivo* data.

**Python Code: in the pythonCode\_neuronSimulations folder there are 4 main simulation files.**

Each file has the changeable parameters:

* durations: array of times, in seconds. A dataset(s) will be generated for each duration.
* parameter\_combinations: array of parameter objects. Currently the only 2 parameters that are being changed are “r” the firing rate in hertz of the poisson spike train input, and using\_sine\_wave, a Boolean determining whether or not the simulation is modeled by the sine wave current that represents the infraslow oscillation.
* Neuron\_num: the number of neurons in the simulation.

Each simulated dataset has 2 “copies.” One copy will end in the suffix \_”groups” and the other will end in the suffix “noGroups.” The dataset pairs are identical, expect the “groups” dataset has three additional neurons that make a contrived assembly with the last neuron in the simulation by having each neuron fire 0.025 seconds after the one before it. For example, if neuron 50 is the last neuron in the simulation, there will be three additional neurons in the dataset: 51, 52, 53. Neuron 51 will always fire 0.025 seconds after 50, 52 will fire 0.025 seconds after 51, 53 will fire 0.025 seconds after 52, etc.

This behavior can be changed at the bottom of the file, starting with the line “spike\_time\_matrix\_seconds\_groups”

The main issue with this code is that it is slow. If using one of the files that generates multiple copies of each dataset, running the simulation will long time durations can take hours and hours. I have worked around this by downloading a “mouse mover” app and running the simulations overnight, but there is significant room for improvement here.

The other things that one might want to change are the directory (folder) path that the data is being saved in, and the naming schema of each file name. Unfortunately, I can’t put these in the changeable parameters section because they key off of variables that change in the loop. However, they can be changed in their locations at ~line 80 of the files.

Currently, all simulated data is stored in the “SimulatedDataSets” folder in the same MaxBluhmSpring2024 folder, with different subfolders.

**Individual files:**

1. **Seeded\_neuron\_simulation\_no\_loop.py: stores data in the “singleDatasetsSeeded” subfolder**
   1. This generates 1 simulated dataset for each specified parameter combination and time duration.
   2. It is seeded in the sense that the datasets for different durations will be identical until the shorter dataset ends. For example, the first 40 seconds of a 160 second dataset will be identical to the 40 second dataset.
   3. Has another changeable parameter, seed\_num, which is the initial seed that is used to generate the data. It can be changed to search for seeds that have strange behavior.

NOTE: I thoroughly commented out this file. I was not as thorough with the others. However, the way the neuron simulation works is the same between all 4 files.

1. **Seeded\_neuron\_simulation\_with\_loop.py: stores data in the “multipleDatasetsSeeded” folder**
   1. This is the same as the seeded\_neuron\_simulation\_no\_loop file, except it generates a number of simulations for each dataset equal to the additional parameter, “num\_simulations.” The seed changes for each simulation number, so the data is different between sim\_x and sim\_y, but remains consistent between different time durations for the same simulation number.
2. **unseeded\_neuron\_simulation\_no\_loop.py: stores data in the “singleDatasetsUnseeded” subfolder**
   1. The same as seeded\_neuron\_simulation\_no\_loop, except it does not seed the data. Meaning data will not remain consistent across simulation iterations or time bins.
3. **Unseeded\_neuron\_simulation\_with\_loop.py: stores data in the multipleDatasetsUnseeded folder.**
   1. Same as seeded\_neuron\_simulation\_with\_loop, except it does not seed the data. Meaning datasets are not consistent across simulation iterations or time bins.

The next piece of functionality that would be good to add is a way of determining synaptic strength between neurons. Unfortunately, I was unable to do this. However, this is how one could add this functionality:  
  
1. Add this code to the changeable parameters section:

connectivity\_matrix = np.zeros((neuron\_num, neuron\_num))

#Each entry [i][j] represents the probability of neuron j firing when neuron #i fires

Then, you can represent neural connectivity like this:

connectivity\_matrix[0, 1] = 0.75 # Neuron 1 has a 75% chance of causing neuron 2 to fire

connectivity\_matrix[0, 2] = 1.0 # Neuron 1 will always cause neuron 3 to fire

Then, after this block:  
  
if voltageVector[S] == Vspike:

spikeVector[S] = 1

counter += 1

voltageVector[S + 1] = Vreset

Add this code:

# Check connectivity matrix to influence other neurons

for target\_neuron in range(neuron\_num):

if connectivity\_matrix[n, target\_neuron] > 0:

if np.random.rand() < connectivity\_matrix[n, target\_neuron]:

spikeVector[S + 1] = 1

This is still a very basic way of defining neural connectivity and I have not had the chance to implement or test this approach. Hypothetically, it should work. Only neurons specified in the connectivity\_matrix will have relationships, all other pairs will be independent. That means if you wanted to create an assembly between neurons 1, 2, and 3, you would only have to define probabilistic relationships between those 3 neurons. You do not have to define relationships between all pairs of neurons.

**Other files in the code base:**

**Isaac Python Code:** My code was based off of work that Isaac Mineo and Professor Jennifer Crondelle worked on. This folder contains Isaac’s original python code for reference. There is more plotting functionality here than I implemented in my code.

**MATLAB CODE: in the matlabCode\_assemblyDetectionScripts**

NOTE: to actually run the detector, these files are stored in the folder:  
Z:\SpecialProjects\DashResearch\InfraslowProject\lizeth\scripts\mBluhm2024

As there are a lot of other files that need to be called to run the detector in the scripts folder. The files in my folder are backups/extras.

**There are 4 main scripts that I wrote. They all share these similar changeable parameters and functionality:**

**Parameters:**

* **dataSeries**: which duration of simulated data are we running the ensemble detector on? This has to be a number that exists in the simulated data.
* **rawpath:** the file path that the simulated data is stored in. Currently, the scripts are pointing towards the seeded data. To point towards the unseeded data, change the "multipleDatasetsSeeded" to "multipleDatasetsUnseeded" and the 'seeded%dSecondData' to 'unseeded%dSecondData' in the rawpath
* **display:** the display parameter for the uonpruned assemblies. You can use “raw’, ‘ordunit’, or clustered
* **criteria:** the pruning criteria. Usually will be set to ‘biggest’
* **BinSizes:** The vector of bin widths to be tested
* **MaxLags:** For each bin size specify the maximum lag lmax (in numbers of bins) to be tested (all lags within [−lmax, lmax ] will be tested):

**Functionality:**Each script runs assembly detection for a dataset (or multiple datasets) and outputs the matlab results in a subfolder of rawpath named ‘output’ and the figures in a subfolder named ‘figures’. If no assemblies are detected, it writes a text file to the output folder with the name of the simulated and the message ‘no assemblies detected’. For more information on how the ensemble detector works, look at the ‘Assembly\_detection\_guidlines’ paper in the ‘papers’ folder of the MaxBluhmSpring2024 folder.

**NOTE**: Unfortunately, you need to manually make the “figures” and “output” folders in the folder where the data is. MATLAB will not make those folders itself

**Differences Between Files:**

**mBluhm\_ensembleDetectorScript\_allFiles.m:**

* This script is designed to run the ensemble detector on all of the files generated by the “no\_loop” python code that generates one copy of each dataset of parameter combinations and time durations.

**mBluhm\_ensembleDetectorScript\_oneFile.m:**

* This script is designed to run the ensemble detector on one of the files generated by the “no\_loop” python code that generates one copy of each dataset for all combinations of parameter combinations and time durations.
* It will list the simulated files in the folder specified by rawpath, and prompt you to choose one to run the detector on.

**multiple\_datasets\_mBluhm\_ensembleDetectorScript\_allFiles.m**

* This script is designed to run the ensemble detector on all the files contained in one parameter combination folder generated by the “loop” python code that makes a number of simulations with different seeds for each combination of parameter combos and time durations.
* It has an additional parameter, ‘parameter\_combo’, that must be specified. The loop generator organizes the data into folders by parameter combo, so which parameter combo you are looking at must be specified.

**multiple\_datasets\_mBluhm\_ensembleDetectorScript\_oneFile.m**

* This script is designed to run the ensemble detector on one of the files contained in one parameter combination folder generated by the “loop” python code that makes a number of simulations with different seeds for each combination of parameter combos and time durations.
* It has an additional parameter, ‘parameter\_combo’, that must be specified. The loop generator organizes the data into folders by parameter combo, so which parameter combo you are looking at must be specified.
* It will list the simulated files in the folder specified by rawpath, and prompt you to choose one to run the detector on.

**Results**

NOTE: assemblies and ensembles are the same thing. I use them kinda interchangeably.

In my preliminary results, the ensemble detector seemed to be working well for our simulated data. However, as we tested more, there were some scenarios where it detected ensembles among 50 completely independent neurons. This is seen when simulations with the “no groups” suffix are detected as having assemblies.

In almost every case with this incorrect ensemble detection, the assemblies were detected at the largest time bin specified in BinSizes.

An early hypothesis is that the ensemble detector was not meant to detect ensembles at the long time scales in our data, but that is not the case. The sample data the Russo and Dorwitz supplied in their tutorial is on the order of 20 minutes. In fact, a current hypothesis is that some of our data is too short, causing ensembles to be incorrectly detected.

In order to explore what could be causing this, I added the seeding feature so that the datasets would be consistent across time bins/simulations in order to rule out possible causes of this strange behavior. We thought random chance might be causing it, so seeding the data generates consistent data to rule out this possibility. I also created simulations that make multiple copies with the same parameters/time durations using different seeds. This is to see if there are certain seeds that cause the strange results. For example, if sim\_0\_seeded\_neuron\_time\_matrix\_120secs\_r=30\_noSine\_noGroups.csv detects as having ensembles, but sim\_1\_seeded\_neuron\_time\_matrix\_120secs\_r=30\_noSine\_noGroups.csv does not, then it is likely that random chance resulting from the sim\_0 seeding caused the strange behavior, not a problem with the parameters or the ensemble detector itself.

If this seeding allows us to determine the cause of the unexpected ensemble detection, then the next step is to implement the synaptic connections component discussed above and run the ensemble detection on those results.

**Next Steps:**

1. Finish running the ensemble detection on the new simulated seeded data.
2. Expand the parameters being used in the ensemble detection scripts. For example, changing time bins, max lags, and detection/pruning criteria
3. Implement synaptic connections.

**ISSUES:**

Hanging up on 320 and 640 second files