Model code is standalone. Plotting code depends on the model code, so they should be decompressed in the same folder.

**Folder Structure:**

* model code: contains the scripts needed to run all the simulations
	+ configuration:
		- configuration.properties: settings for running the model:
			* IS\_MATLAB\_STANDALONE: this is set to true if matlab code is compiled (e.g. on a cluster) - with the compiled file the same name as the .m file
			* IS\_BRIAN\_STANDALONE - this is set to true if brian code is to be run in standalone mode (c compilation)
			* DATA\_FILE\_PATH: where to store the simulations data
			* TMP\_FILE\_PATH: where to save temporary simulations file (compiled code needed at runtime, log files)
		- configuration.py: loads the configuration into a python dictionary
	+ data\_analysis: code for analysing model time series (Kappa, DFA, fEI, power, PLF)
		- make sure to add all subfolders in the data\_analysis folder to the matlab code before running the code (e.g. *addpath(genpath(pwd)* while in the folder)
		- run\_analysis.m: main entry point for analysis, input parameter is the directory that contains the simulation data
		- batch\_run\_analysis.m : script to run analysis in parallel (multiple cores) in a folder with multiple simulations
		- compute\_dynamicrange\_plf.m: computes dynamic range for tables of simulation data
		- aggregate\_plf\_data.m: input is path to folder with multiple simulations, the script aggregates the stimulation analysis data across all these simulations, for use by the “fig\_corr\_LRTC\_vs\_versatility\_1S2\_2S2\_3S2\_1F\_2F\_3C\_alldata.m” script in the “plotting” folder
		- calculate\_baseline\_plf\_bootstrap.m: calculates the 95% percentile of PLF for trials in a pre-stimulus ampltitude bin for a 2000 seconds stimulation
		- calculate\_baseline\_phaseregulation\_bootstrap.m: calculates the 95% percentile of phase regulation when averaged across 10 runs of 2000 seconds
	+ parameter\_files: contains two types of files – \*.Parameters files – which specify the parameters for running individual simulations, and \*.Batch files – which contain all parameter sets used for our simulations – the parameter set on each line is interpreted with the “Simulation parameters” in the \*.Parameters file that the batch is run with.
		- NetworkGen.Parameters: parameters for generating an individual network, all parameters sets used: ParamspaceNetworkGen.Batch
		- StimulationModelExcWeight: parameters for two variants of model with different excitatory to excitatory weights, all parameter sets used: ParamspaceExcWeight.Batch
		- StimulationModel.Parameters: parameters for running an individual simulation, all parameter sets used: ParamspaceStimulus.Batch
		- StimulationModelOutputSpikes.Parameters: same as above, but also outputs individual spikes for all neurons
		- StimulationModelInputWeight.Parameters: parameters for running individual simulations, allowing for variation in the input strength, all parameter sets used: ParamspaceStimulusInputWeight.Batch
	+ utils: utility functions for data analysis, parameter loading, network generation, data output
	+ main\_network\_generation.py: python script for generating and saving a network structure, receives the following parameters as input - first the parameter file with the generic parameters for generating a network (NetworkGen.Parameters), and after that, the specific parameters for generating this individual network (which are to be interpreted by the “Simulation parameters” property from parameter\_files/NetworkGen.Parameters). So, since Simulation parameters=Run Number,Excitatory Connectivity,Inhibitory Connectivity, then running *python main\_network\_generation.py NetworkGen.Parameters 1 0.50 0.80* will generate a network with Run Number 1, Excitatory Connectivity 50% and Inhibitory Connectivity 80%. The location where the network is saved is specified under the Output dir path format parameter of the parameter\_files/NetworkGen.Parameters file.
	+ main\_sim\_stimulation.py: python script for running a simulation, receives the following parameters as input – first the parameter file with the generic parameters for running a simulation (e.g. StimulationModel.Parameters), and after that, the specific parameters for running a simulation (which are to be interpreted by the “Simulation parameters” from StimulationModel.Parameters). So, since, Simulation parameters=Run Number,Excitatory Connectivity,Inhibitory Connectivity,Stimulus size, runing *python main\_sim\_stimulation.py StimulationModel.Parameters 1 0.60 0.75 5* will set Run Number to 1, Excitatory Connectivity to 0.6, Inhibitory Connectivity to 0.75, and Stimulus size to 5. Simulations will be saved under to the data path specified in configuration/configuration/properties, at a location specified by *StimulationModel.Parameters* → *Output dir path format*. The simulations will load existing networks as specified under *Network to load path format*, and *Network to load path input parameters* from *StimulationModel.Parameters*. If you want to run a simulation without loading an existing network, you can just comment out the *Network to load…* lines from the Parameters file (to comment a line use the sharp ‘#’ character at the beginning of the line).
	+ batch\_run.py: script for running on multiple cores a batch of parameters. Opens a number of threads equal to the number of virtual processors on the computer minus 1. To run network generation for the entire batch of networks, use *python batch\_run.py NetworkGen.Parameters ParamspaceNetworkGen.Batch*. To run all stimulation model simulations, run *python batch\_run.py StimulationModel.Parameters ParamspaceStimulus.Batch*. Console output should show progress in running through the batch files. To troubleshoot, go to the temporary folder (see configuration/configuration.properties file), and under the log folder, there will be a log file generated for each line in the batch file.
	+ generate\_network.params.py and the other generate\*.py files were used to generate the batch files: *ParamspaceNetworkGen.Batch* and the other *Paramspace\*.Batch*
	+ build\_results\_table.py: once all the simulations have been run, use this script to aggregate all simulation results. Run it using as parameter the folder path in which all simulations have been saved.
	+ requirements.txt: all python packages required to run the model
* plotting: contains all the scripts needed to make the figures in the paper:
	+ to generate all figures run the make\_all\_figures.m and make\_all\_figures.R scripts
	+ note that fig\_correlation\_LRTC\_vs\_dynamicrange\_and\_prestimreg\_all\_data.m, fig\_rasterplots\_alldata.m and fig\_raw\_signal\_phases\_alldata.m also contain a nodata variant. If the data is not available, you can generate it yourself by running the code, or use the nodata variant.
	+ The scripts fig\_plf\_1C\_2AC, fig\_power\_kappa\_dfa\_3nets\_1S1DFH, fig\_prestim\_regulation\_2D\_3A, and fig\_raw\_signal\_phases\_3S1\_alldata will produce slightly different plots across reruns, because of the random noise which is added to the signal at each execution of the scripts
	+ select\_net.m contains the criteria used for selecting the subcritical/critical/supercritical networks used throughout the paper
	+ configuration.properties file contains the path where datasets are stored, model scripts for analyzing the data, as well as where to save the figures. As long as the model code, plotting and simulations data folders are in the same subfolder, the current options should work.
* simulations data: contains the csv files with all the simulations data; simulation runs with the network structure stored, as well as the time series; and also intermediary data used to make figures in the cases where the entire data could not be included due to space constraints.

**How to run the model simulations**

**Software requirements: python ( tested with python 2.7.3 and 3.5.9 ), matlab, R (tested 3.6.3)**

**Folder for this part: *model* *code***

**Before running the model:**

- install all python requirements using: *pip install -r requirements.txt* ***(if you run into any problems with the dependencies, then install, line by line, each dependency)***

- modify the paths in the *configuration/configuration.properties* file to reflect the locations where you will want to store the data on your computer, the location/name of the python executable, or whether you want to use compiled matlab code.

- for the simulations, the code can be run in python mode or in standalone mode. Standalone mode is much faster, since it compiles python code to c code. This should work out of the box on linux (tested on ubuntu), but on windows it may require additional configuration (see brian2 documentation website <https://brian2.readthedocs.io/en/stable/>). If you are having trouble linking the compiler to python, on windows, you can also turn the standalone mode off, by setting *IS\_BRIAN\_STANDALONE* to False, in the *configuration.properties* file.

**Simulation analysis**

Code is in the *data\_analysis* folder. Run the *run\_analysis* function, which takes as first parameter the path of a simulation folder: e.g. *run\_analysis* ‘*path\_to\_data/EC0.60.IC0.75.Stim5.Run1/*’

**Network initialization**

To ensure that simulations for different stimulus sizes are run for the same network initializations, we initialize all networks from the beginning, before running the simulations. This can be done by running *python* *batch\_run.py main\_network\_generation.py NetworkGen.Parameters ParamspaceNetworkGen.Batch*

where:

*NetworkGen.Parameters* – contains the parameters used for network generation

*ParamspaceNetworkGen.Batch* – all parameter combinations. Each line is interpreted according to the *Simulation parameters* property in *NetworkGen.Parameters*

Networks are saved under the path specified in *configuration/configuration.properties* – DATA\_FILE\_PATH, in a subfolder whose name and format is determined by the *Output dir format* property of *NetworkGen.Parameters.*

**Content of each network folder**

Simulation.Parameters – parameters used to generate the network

SimulationData.h5 – hdf file which contains under *networks/main\_network/types* the vector of types for all neurons on the grid (excitatory = 1, inhibitory = 0). *connections/from\_main\_to\_main/indexes* contains the index of all pre- and post- neurons for all synapses. *connections/from\_main\_to\_main/weights* contains all synaptic weights.

output.txt – the simulation output – can be checked to see if there were any errors while running the network generation

**Stimulation model runs**

All at once:

- after all networks are generated you can run all the simulations using python *batch\_run.py main\_sim\_stimulation.py StimulationModel.Parameters ParamspaceStimulus.Batch*

*-* alternatively this can be run on a cluster, where each line in *ParamspaceStimulus.Batch* is submitted to the processing queue. If the cluster requires compilation of matlab code, then compile the code in the data\_analysis folder, and set IS\_MATLAB\_STANDALONE to True in the *cros\_model/configuration/configuration.properties* file

One simulation at a time:

- if you want to run just one simulation (say 60% excitatory connectivity, 75% inhibitory connectivity, run number 1, stimulus size 5), then first make sure that the network is initialized (if you followed the step for network generation no need to do anything here. alternatively, you can just generate this one network by running *python* *main\_network\_generation.py NetworkGen.parameters 1 0.60 0.75*), and then run *python main\_sim\_stimulation.py StimulationModel.Parameters 1 0.60 0.75 5*.

Simulation data is saved under the path specified in configuration/configuration.properties – DATA\_FILE\_PATH, in a subfolder whose name and format is determined by the Output dir format property of  *StimulationModel.Parameters*

**Content of each simulation**

For each simulation, the following files will be saved:

Simulation.Parameters – all parameters used to run the simulation

SimulationData.h5 – hdf file which contains under *data/main\_network\_excitatory\_spikes\_timeseries* the time series with the total number of excitatory spikes in the network, for each timestep. Under *data/main\_network\_inhibitory\_spikes\_timeseries* the same information, but for inhibitory neurons. Under data/*external\_input\_spike\_times* the timepoints where the network was stimulated (indexing is 0-based). If Output activity flag is set to True in the Parameters file, then individual spike data will be saved for all neurons. For excitatory neurons: data/*main\_network\_excitatory\_spikes\_individual\_neurons* (index of neurons which spiked) and data/*main\_network\_excitatory\_spikes\_individual\_times* (index of times when they spiked), for inhibitory neurons, the same but with *data/main\_network\_inhibitory\_spikes\_individual\_neurons* and *data/main\_network\_inhibitory\_spikes\_individual\_times*. Note that the excitatory/inhibitory indexes are the ordered indices of these neurons within the respective excitatory/inhibitory populations. For the exact positioning of these neurons on the network see in the network file *networks/main\_network/types.*

output.txt – the simulation output – can be checked to see if there were any errors while running the simulation

ActivityStatistics.txt – contains all statistics run on the time series (DFA, fEI, Kappa etc) – this is output by the *run\_analysis* file in the data\_analysis folder

**Aggregate data**

After you have run all the simulations, you can run the build\_results\_table.py script, which takes as input parameter the path to the folder where all the simulations are saved. You can then use the resulting csv table, and the plotting code (below) to make the plots.

From data\_analysis, run aggregate\_plf\_data on the folder with the simulations to aggregate all plf and prestimulus regulation data for timepoints around stimulation.

From data\_analysis, run compute\_dynamicrange\_plf (make sure to update the paths to the csv file that came out of build\_results\_table.py) to compute the dynamic range of plf values.

**How to replicate the figures in the paper**

**Software requirements: MATLAB, R**

**Folder for this part: *plotting***

For R (tested with 3.6.3), make sure to install the packages *ggplot2*, *dplyr* and *scales*. (in R, issue the command *install.packages(“ggplot2”)* …).

To produce the figures, run these two scripts:

- make\_all\_figures.m (MATLAB)

- make\_all\_figures.R (R)