**Cortical representation of touch *in silico -* NetPyNE implementation**

**Loading network data files from Matlab**

This initial NetPyNE implementation requires loading data from a network instantiation of the existing Matlab model version. Multiple model instantiations (realizations) with different connectivity randomization seeds are available at this repository: <https://data.donders.ru.nl/collections/mine?1>. The data includes two different model architectures: 1 barrel, and 3 barrels (one principal, two secondary). Within these instantiations, there are two files necessary to run the NetPyNE-based simulations, for example: “CMDMs\_forNetPyNe1.mat” and “CMDMs\_forNetPyNe1\_ConData.mat”. These files include detailed information about barrel structure, cells (properties and location), recurrent cortical connections (ids of connected neurons and properties of the connections), and thalamic-to-cortical cells’ connections.

**Setting and running simulations based in NetPyNE**

All commands to generate and run the model in NetPyNE are included in “init.py”. Here, we review, in order of execution, all the relevant details, including the different files and functions used and how they relate to the original model components and functionality.

* ***settings.py***

The method setup() in “settings.py” sets the following definitions:

* + **Input Option*:*** ‘Multitrial’ or ‘Svoboda’.
    - Multitrial: It corresponds to reading the thalamic input as a PSTH measured in advance under controlled whisker movements. For example, as in the work [“Spatiotemporal gating of sensory inputs in thalamus during quiescent and activated states” by Aguilar & Castro-Alamancos, J. Neurosci. 2005]. The relevant measurements from this study are given (file “psth.dat” in the folder “ReadingData\_Aguilar”).
    - Svoboda: It corresponds to the structure provided by the matlab code when reading real whisker movements from a freely available dataset from Svoboda lab (<https://crcns.org/data-sets/ssc/ssc-2/about-ssc-2>) and applying a barreloid model, where these inputs are filtered with proper kernels.
  + **Model:** A dictionary providing the location (folder) and the aforementioned files with the model instantiation.
  + **ExperimConds:** Dictionary with the properties corresponding to the simulation settings, the number of trials (inputs changes as the trials does) and the number of repetitions (thalamic spike train inputs are frozen, but stochastic effects appear due to internal noise).
  + **Settings:** Dictionary with some properties defining the simulation:
    - vr: Resting potential
    - v0: Initial potential
    - u0: Initial recovery variable potential
    - dyn\_thres: 0 corresponds to the classical Izhikevich model, whereas 1 to the adaptive one (see “Spike-threshold adaptation predicted by membrane potential dynamics in vivo” by Fontaine, Peña & Brette, PLoS Comp. Biol. 2015).
    - tau\_plas: The time constant of the short-term plasticity
    - fr: Fraction of recurrent connections implemented in the model, from the total number of connections provided by the matlab instantiation. This is useful to develop new features, and test them fast.
* ***reading.py***

Includes two methods, one to read the thalamic inputs, and the other to read the instantiated model from Matlab.

* + The method read\_input(...) in “reading.py” uses the Input[‘Option’] defined in “settings” to read either the PSTH (“Multitrial”) or directly the instantiated spike trains (“Svoboda”)
  + The method read\_model(…) reads all the information in the files provided by Matlab with the instantiated model. At this stage, the model preserves the Matlab structure.
* ***assemble.py***

Includes three methods that restructure the network instantiated in Matlab to be usable by NetPyNE, and read in the previous data as dictionaries:

* + **define\_barrels(…)** creates the dictionary “Barrels” with all the relevant information. Importantly, the barrels are read as one-dimensional entities (i.e. number of barrels in y-direction is 1). However, this can be readily extended to two-dimensional barrels, according to the information (and the structure) of the Matlab output. Also, the number of barrels and the number of thalamic cells are returned.
  + **define\_pops(…)** creates a dictionary “Pops” with all the cortical populations involved in the network, with detailed information in an individual basis (i.e. cell by cell) for relevant properties: position and properties of the intrinsic dynamics of individual cells. These later properties are collected in a ‘params’ dictionary-entry, to be directly used in the NetPyNE specification of populations. Also, the number of cortical cells (Ncells), the number of populations, even empty but defined in the matlab structure (Npops), and a list of ids labeling cells belonging to a given population (N). These ids are local to the cortical network (thalamic cells are not included) and start from 0, as python listing does.
  + **define\_conns(…)** creates three dictionaries with all the specifications of given connections: thalamus to cortex, excitatory recurrent connections in S1 (within and across barrels), and inhibitory recurrent connections in S1 (also within and across barrels). Each entry in these dictionaries corresponds to a relevant property for the connections (ids of pre/post, and specific properties for the connection dynamics itself), and is stored as a list of property values (one for each connection).
* ***netParams.py***

The method set\_netParams(…) uses the dictionaries representing all the network structure to set the network parameters, with most of the information inherited from Matlab.

* **Cells:** The Izhikevich model implementation is described below in more detail. A generic cell section is created to host the Izhikevich model, modeling all cortical cells. These cells have geometry in the NEURON framework (compartCell class in NetPyNE), so they have geometric properties. As mentioned in the NEURON portal[[1]](#footnote-1), for single compartment simulations it is most convenient to choose a membrane area of 100 micron2 so that point process currents (nanoamps) are equivalent to density currents (milliamps/cm2). This has consequences on the scale of synaptic amplitudes set in the present program (a factor of 0.001, see corresponding dictionaries in define\_conns(…) in assemble.py).
* **Populations:** Once cellular properties are set, populations can be defined. We first create a population called “artificial” corresponding to the **thalamic inputs**. Each cell in this population produces a spike train with specific times, which may originate in two ways: 1) externally loaded from a model (“Svoboda” option), 2) generated as Poissonian spike trains from controlled experiments in the thalamus where the Psth were measured/calculated (“Multitrial” option). Cellular locations associated to this population, which are irrelevant, are set in a very narrow spatial domain at the origin. To set these spike times, a specific object called “VecStim” is used (see nonlinear mechanisms).

Next, all **cortical populations** are created based on the aforementioned cellular specification (IzhiCell in cellParams) via the popParams dictionary. Labels of these populations are coincident to those in matlab (from “1” to “15”: “1”-“4” corresponding to layer 4, “5”-“15” corresponding to layer 2/3). Individual cellular properties are loaded with a specific “cellsList”, with all necessary individual properties: locations, not very important now that the connections are already established in the matlab structure, and properties relative to the intrinsic dynamics within the dictionary “params” (see define\_pops(…) in assemble.py). Worthy to note is that regarding the locations, in NetPyNE it is customary to set the depth as the y-coordinate i.e. 0 at the top (pia) and max y at the bottom (white matter), so this was taken into account when reading individual positions.

* **Connections:** Once the populations are set, individual connections can be established. Before, a generic synaptic mechanism is set with somewhat arbitrary properties, based on the nonlinear mechanism included in “FluctExp2Syn.mod” (see nonlinear mechanisms). All connections are read from the “ConData” structure inherited from matlab. Identity of pre- and post-synaptic cells, as well as the mean amplitude and the delay, are individually set in the “connParams” dictionary in NetPyNE. Here, the individual identifier (id) of each cell is RELATIVE to the “conds” in the rule, which in this case coincide to the identifier from matlab (minus 1, since in Python everything starts from 0). Other parameters characterizing the synapses of individual connections are set with arbitrary values (default values in the .mod file and/or arbitrary values in the synMechParams definition), except from the type (excitatory/inhibitory). Three sets of connections are defined based on the dictionaries returned in define\_conns(…) in assemble.py. In this way, all individual (and identified) connections are established, although some specific properties then need to be updated at the level of individual synapses.

**Nonlinear mechanisms included in the model**

*Single cell dynamics – Implementation of the Izhikevich model*: The dynamics of each cell in the cortical network is based on a modified version of the Izhikevich model, with a history-dependent threshold (dyn\_thres = 1). These dynamics are specified in a .mod file, based on a previous implementation of the Izhikevich model (<https://senselab.med.yale.edu/ModelDB/showmodel.cshtml?model=39948>). The new file, “izhi2007b\_dyn\_thr.mod”, includes the dynamics of the threshold (for certain types of cells, “celltype<5”, see COMMENTS section), and the interface with the hoc interpreter to set all properties and communications. Three points are worthy to note:

1. The Izhikevich model has a voltage level at which a spike is declared and the reset rule is applied. This voltage level is set by the parameter “vpeak”, here set at 10 mV. Once this voltage is reached, the reset rule inside the .mod program is applied. However, the voltage update is not performed in the .mod file, but in the NEURON simulator. So, a different (lower) “monitor” threshold is set in this program (via NetPyNE, netParams.defaultThreshold = 0.0, in cfg.py) to establish when a spike occurs, so all associated spiking events are sent to the NetCons (objects that perform the synaptic communications). This monitor threshold is set higher enough so false positives are not allowed, but lower enough so time discretization enables this to be reached and not skipped in the .mod calculation (otherwise, a finer time discretization may be necessary). Take into account that the difference between this “monitor” threshold for declaring the communication event and the threshold used to apply the reset rule results in an effective delay, that eventually can be evaluated.
2. The initial condition is set in the NET\_RECEIVE block. It could be vr (rest potential), but any other specified via a new parameter. In this implementation, it is set by “v0” so it may be different from the resting potential.
3. In multitrial experiments, a given initial condition is assumed before a stimulus is presented. The dynamical voltage threshold in the adaptive Izhikevich model, vt, is set to its stationary value at the initial potential, vt = vt\_thetainf(v\_0).

* *Thalamic inputs*

Thalamic inputs are set as a population of a specific object called “VecStim”, that also has a .mod file associated. This object delivers spikes at specific times.

* *Synapses*

The synaptic mechanism is coded in a specific .mod file, “FluctExp2Syn.mod”, which includes the dynamics of the gating variable, the stochastic nature of the effective transmission (failure rate), the stochastic nature of the amplitude at the postsynaptic side, and short-term learning dynamics. Also, there is commented block NET\_RECEIVE which includes a “flag\_print” to monitor how things are calculated, useful during development stages.

* *NMOD file compilation*

A final reminder: Here we have three .mod files (for the dynamics of individual cells, for the synapses, and for the input spike trains). BEFORE running the NetPyNE code (and even before starting an IDE, if the program will be run from there), the .mod filed should be compiled. That is, from a terminal (or command window), setting the current directory to the location of the program, it has to be executed “nrnivmodl”.

* ***cfg.py***

The method set\_cfg(…) **sets all configuration options** for NetPyNE. In particular, if inputs are “Multitrial”/”Svoboda”, total time is set to 40/6000, respectively. This parameter has to agree to the definition of the inputs (checked by the user).

Some traces are defined to be recorded, as an example of the capabilities of NetPyNE (see recordTraces), with different configuration options.

Simulation results are stored as .json files, with only simData (here, it corresponds to the spike times, the spike ids, and recorded traces from identified cells –set by the plotting options-).

The raster plot and traces for specific cells are plotted. These cells are one per population, in the main barrel. Ids has been identified in advance.

* ***sim.create(…)***

With this command **the complete network is instantiated**, although with some arbitrary values in the individual connections and in the timing of input spike trains. Once finished, all NEURON objects have been created and therefore, are available to update.

* ***netModify.py***

The method update\_net(…) is used to **update connections**. Once the connections are instantiated with the sim.create(…) command, the associated individual objects (NetCon, accessed through the corresponding “hObj” in NetPyNE objects) are modified to specify individual synaptic characteristics for each connection. This is done with this method, taking into account a possible distributed parallel computing. Before updating, all (post-synaptic) cells associated to a given node are determined, and then updated accordingly. Worthy to note is that, at this point, for each connection the identity of each pre/post cell is via the “global” id, so the global and the relative identifier has to be disentangled (necessary to load the properties). In the present version, the thalamic population was defined first (see netParams.py, then NetPyNE creates cells in the order given by the order of the populations), so all thalamic cells have the first ids, and then the cortical cells are assigned. Properties of individual connections were already defined by the dictionaries in define\_conns(…) in assemble.py, so here the update is straightforward.

* ***multitrial.py***

The method run\_multitrial(…) sets a **multitrial paradigm** (different spike trains from the same ensemble), with repetitions (the same spike train several times, to study the role of internal noisy processes on individual spike trains). This is set once the network is completely defined, via **modification of the NEURON object implementing the VecStim** nonlinear mechanism corresponding to spike trains.

When the “Svoboda” option is selected, spike trains are directly loaded so the number of trials is given in advance. This may or may not coincide with the number of trials defined in “settings.py”. If the available number of real spike trains is less than the number of trials specified in the settings, random selections within the given set are considered. Again, spike times are set via the NEURON object associated to the VecStim nonlinear mechanism.

Each trial is saved in json files for posterior offline analysis, although the suite NetPyNE can be used for in situ analysis. For example, in each trial, the rasterPlot and some traces are recorded (set in the simCfg.py file)

### Multitrial Experiment Results

#### Description of model populations (for 3 barrels)

Neuron indices for each populations (Principal barrel=magenta ids; Yellow ids: Secondary barrels) :

0-599: [0-199; 200-399; 400-599] Thalamic spike trains

600-1919: [600-1039; 1040-1479; 1480-1919] 1320 cells Pyr L4

1920-4721: [1920-2853; 2854-3787; 3788-4721] 2802 cells Pyr L4

4722-5003: [4722-4815; 4816-4909; 4910-5003] 282 cells Inh L4

5004-5282: [5004-5096; 5097-5189; 5190-5282] 279 cells Inh L4

5283-11978: [5283-7514; 7515-9746; 9747-11978] 6696 cells Pyr L2/3

11979-12296: [11979-12084; 12085-12190; 12191-12296] 318 cells Inh\_FSBS L2/3

In each pop, the first and the third lists correspond to secondary barrels, the middle one is principal

12297-12308: [12297-12300; 12301-12304; 12305-12308] 12 cells Inh\_FSCH L2/3

12309-12473: [12309-12363; 12364-12418; 12419-12473] 165 cells Inh\_BSPV L2/3

12474-12665: [12474-12537; 12538-12601; 12602-12665] 192 cells Inh\_Mar L2/3

12665-12665: [] 0 cells Inh\_Bit L2/3

12666-12857: [12666-12729; 12730-12793; 12794-12857] 192 cells Inh\_DBC L2/3

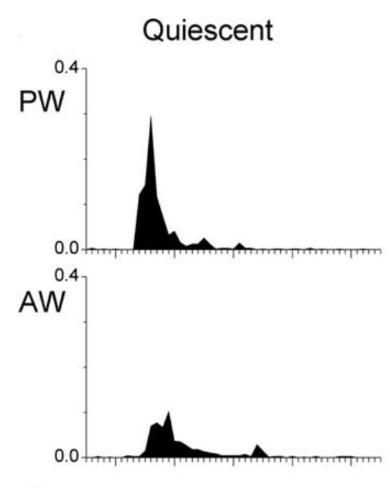
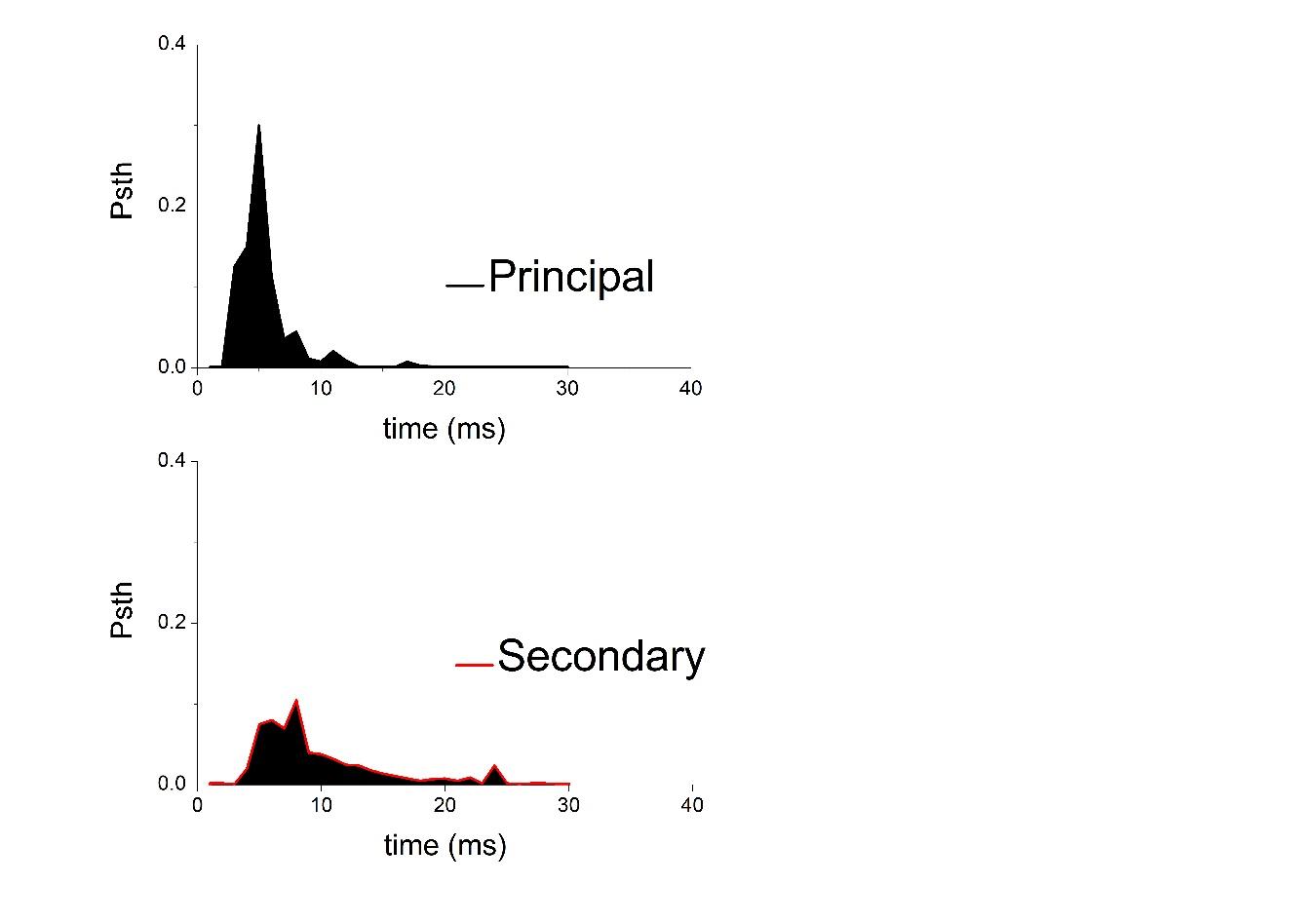
12857-12857: [] 0 cells Inh\_Bip L2/3

12858-12959: [12858-12891; 12892-12925; 12926-12959] 102 cells Inh\_Bip L2/3

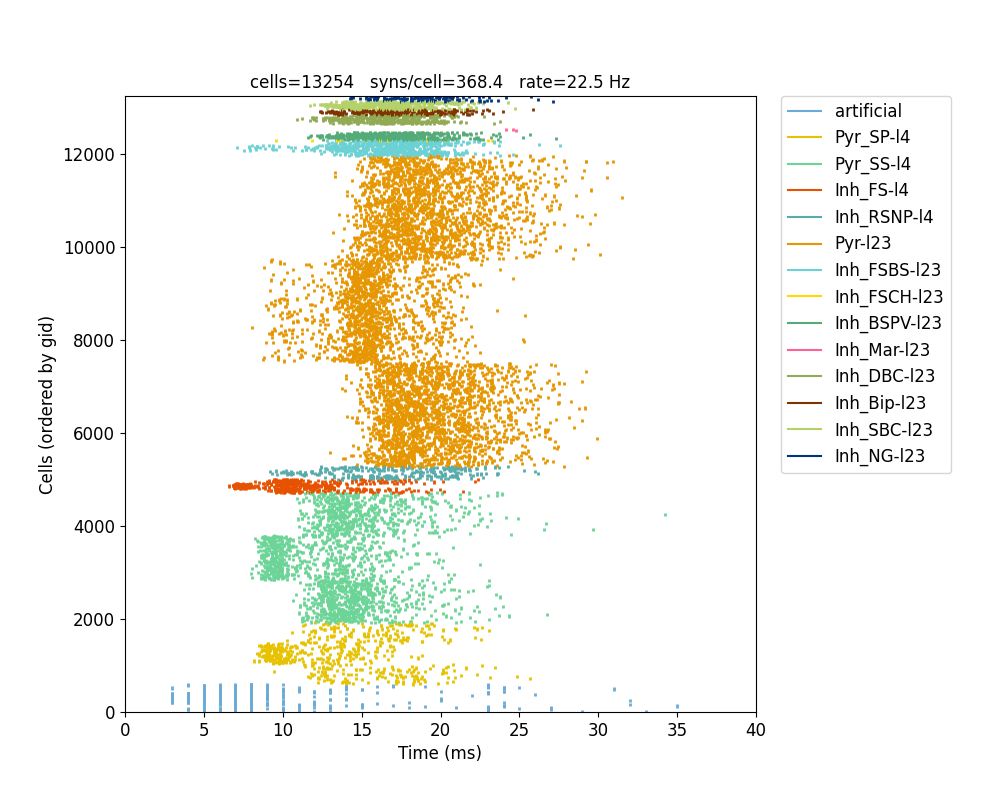
12960-13139: [12960-13019; 13020-13079; 13080-13139] 180 cells Inh\_SBC L2/3

13140-12653: [13140-13177; 13178-13215; 13216-12653] 114 cells Inh\_NG L2/3

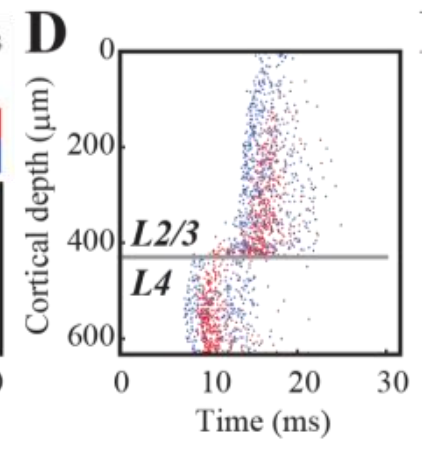
#### PSTH of thalamic input spike trains

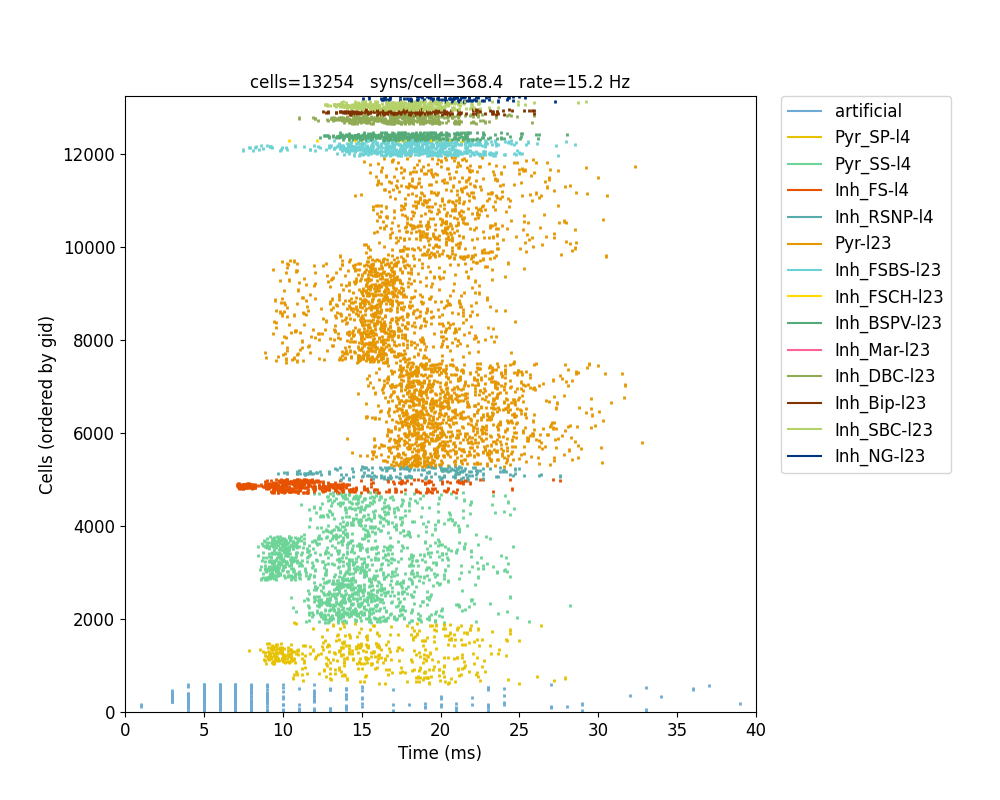
Thalamic input spike trains for multitrial experiment defined based on from Aguilar and Castros-Alamancos, 2005 (J Nsci) data (Fig shows spike probability / 1-ms bin; *left*: paper fig; *right*: digitalized version): 

#### Raster plots

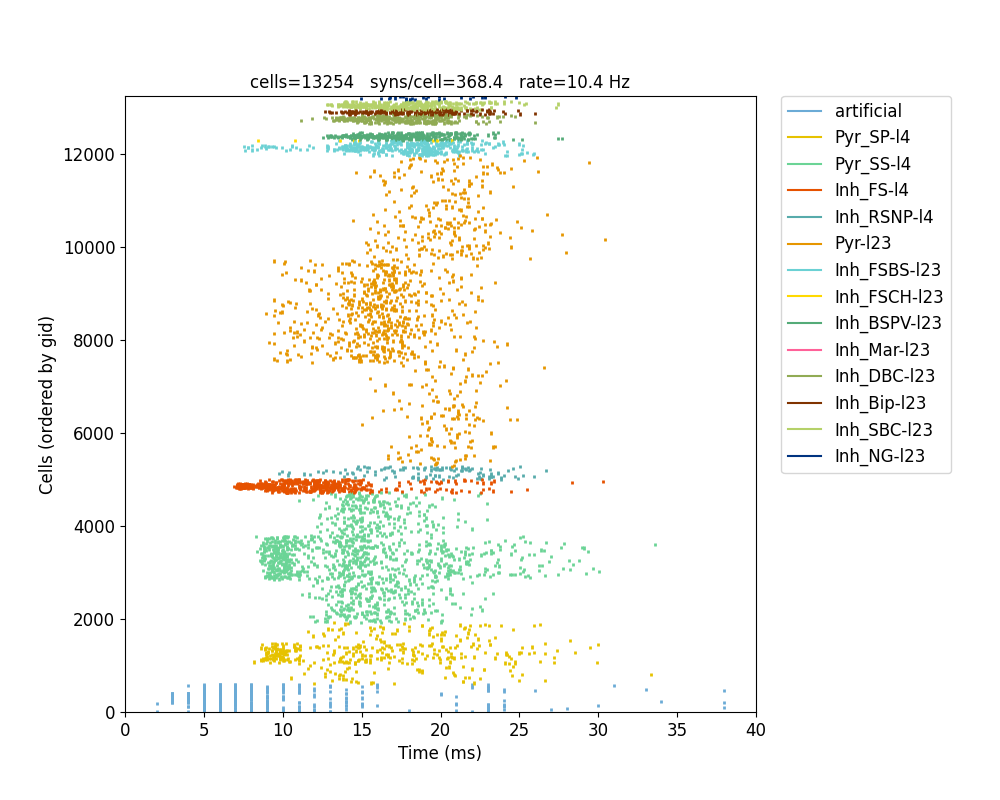


Vr = -60 mV





Vr = -70 mV



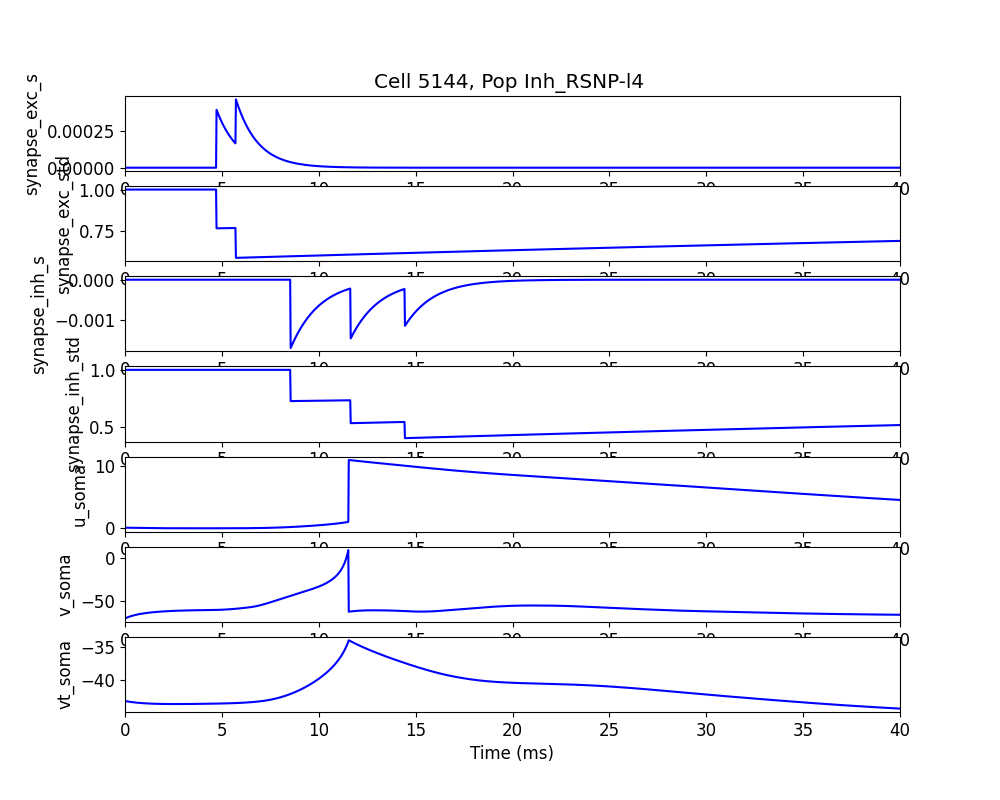
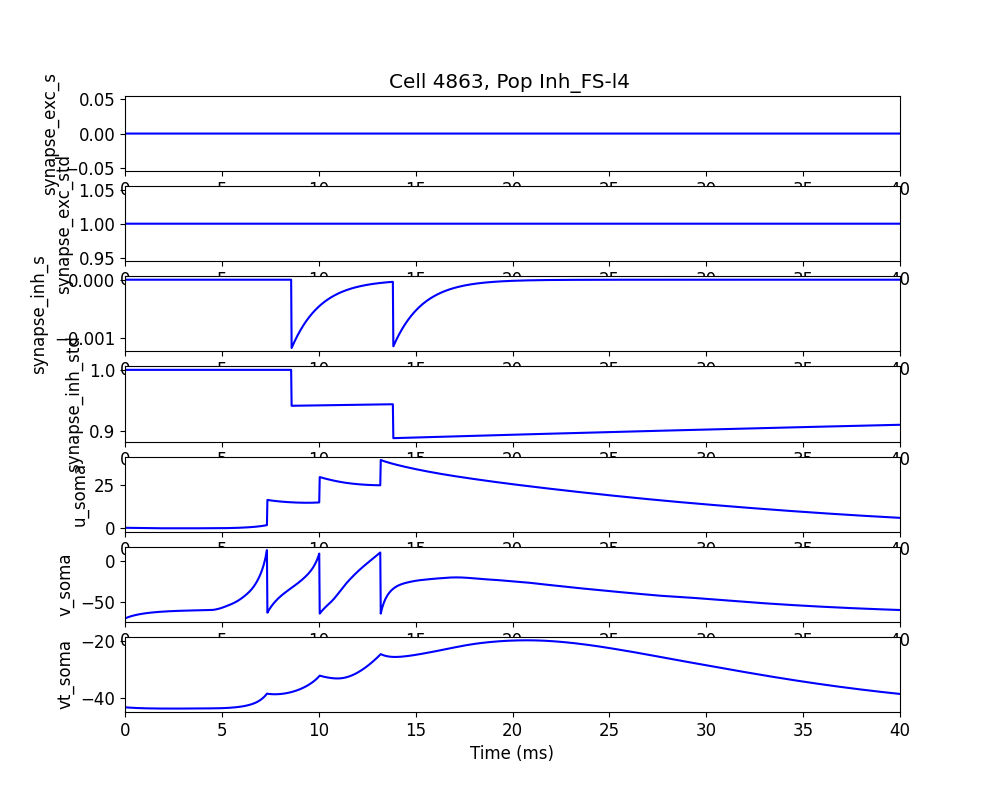
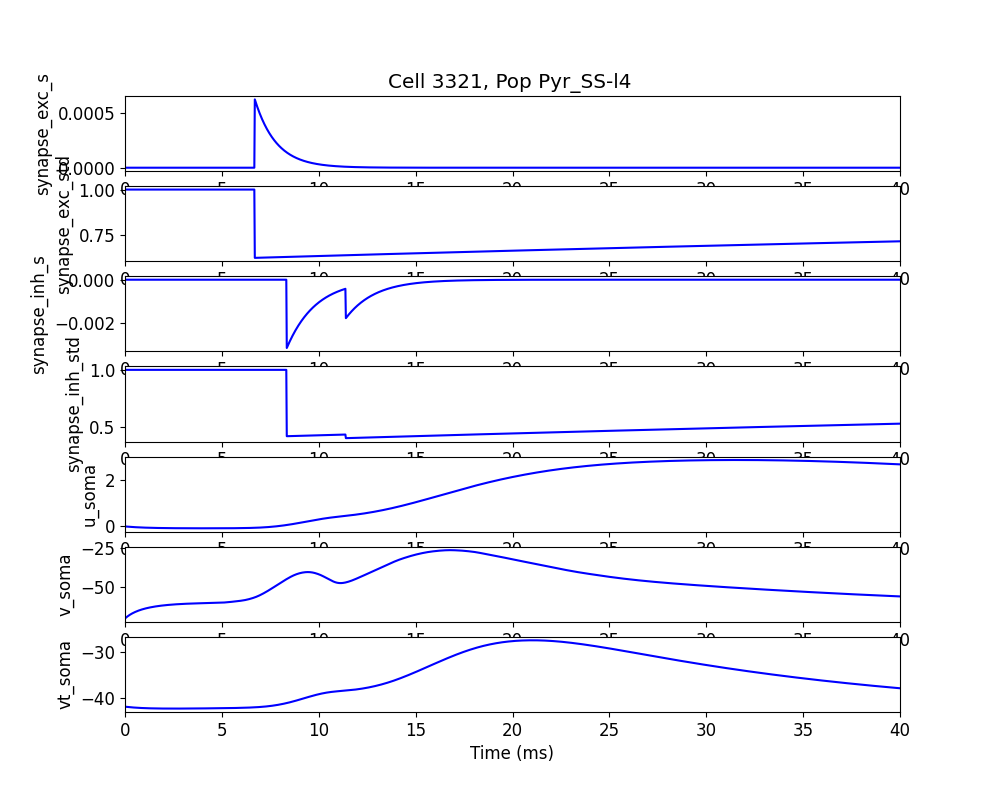
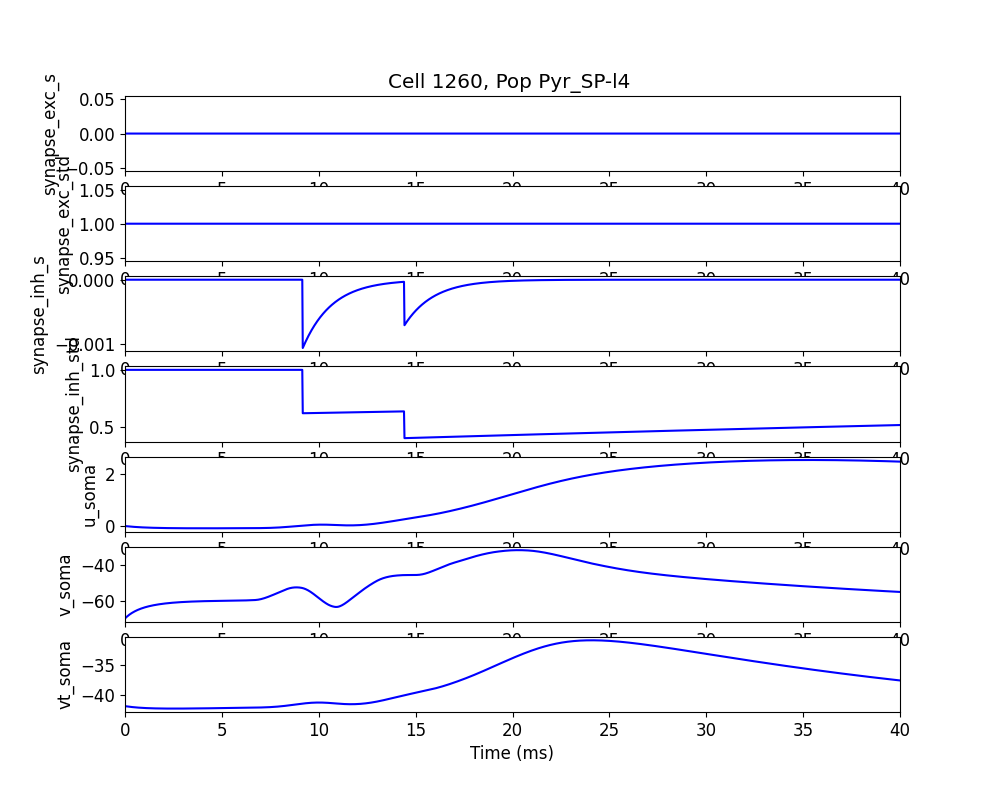
Vr = -80 mV

#### 

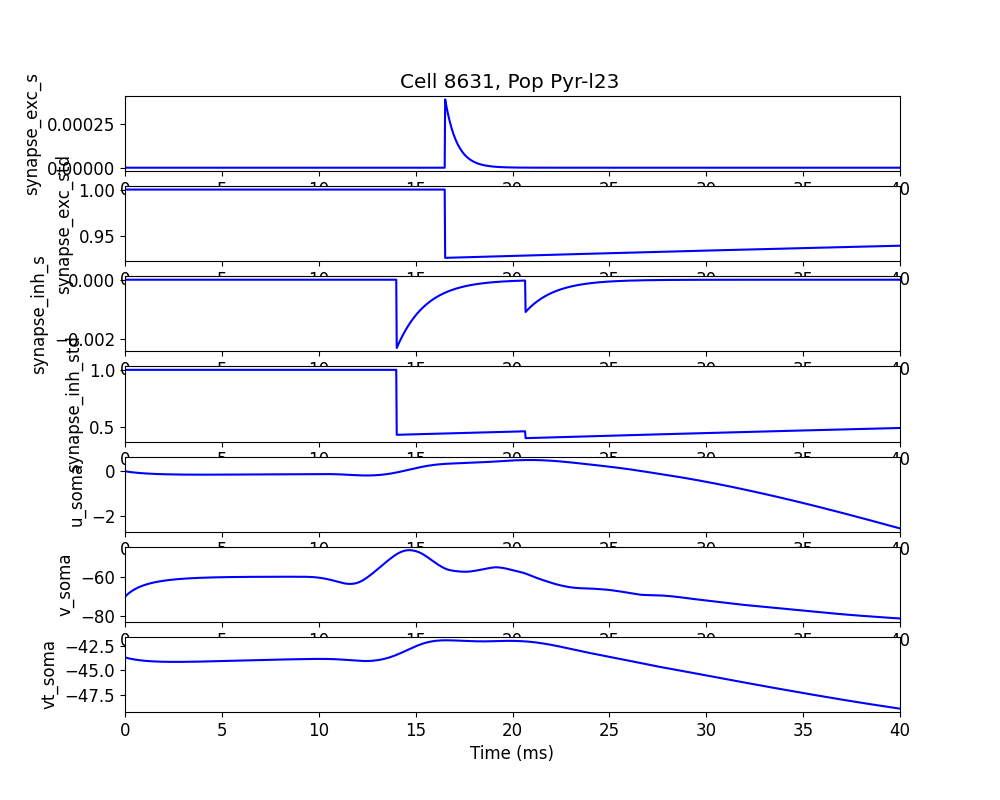
#### Example voltage traces (from main barrel)

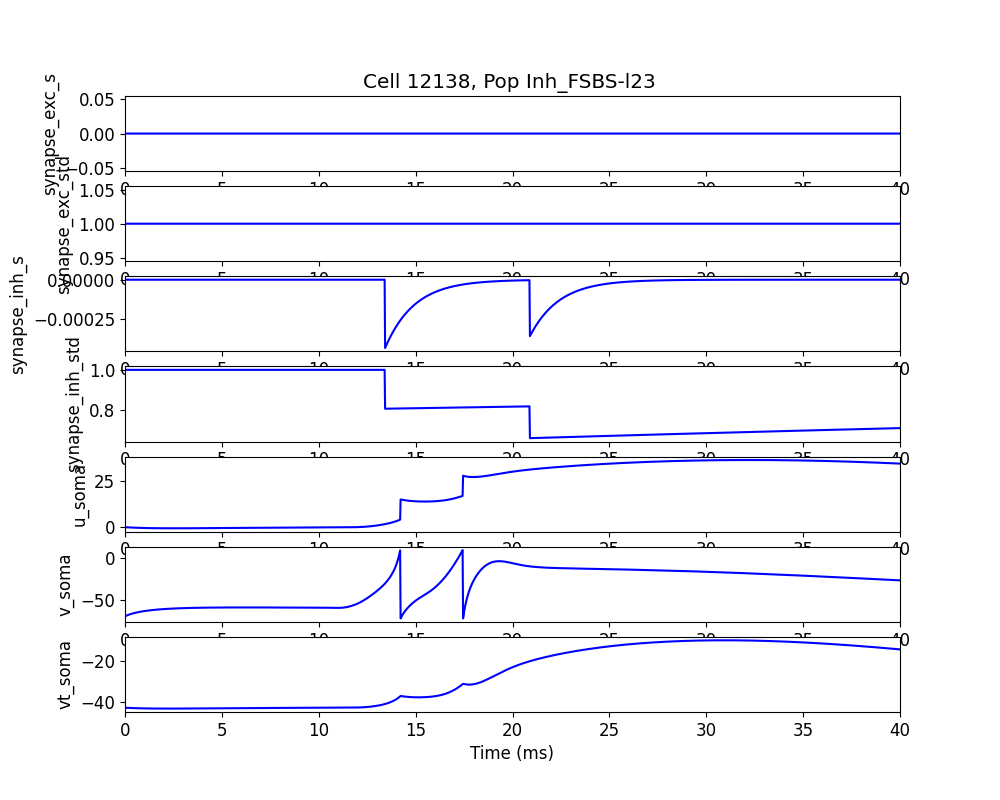
In this case, initial potential was v0 = -70mV, whereas vr = -60 mV. First four traces correspond to: the gating variable “s” and the short-term dynamics factor “std” of an excitatory synapse impinging on this (post-synaptic) cell, and the same for an inhibitory connection. The last three traces correspond to the dynamical variables in the threshold adaptive Izhikevich model.

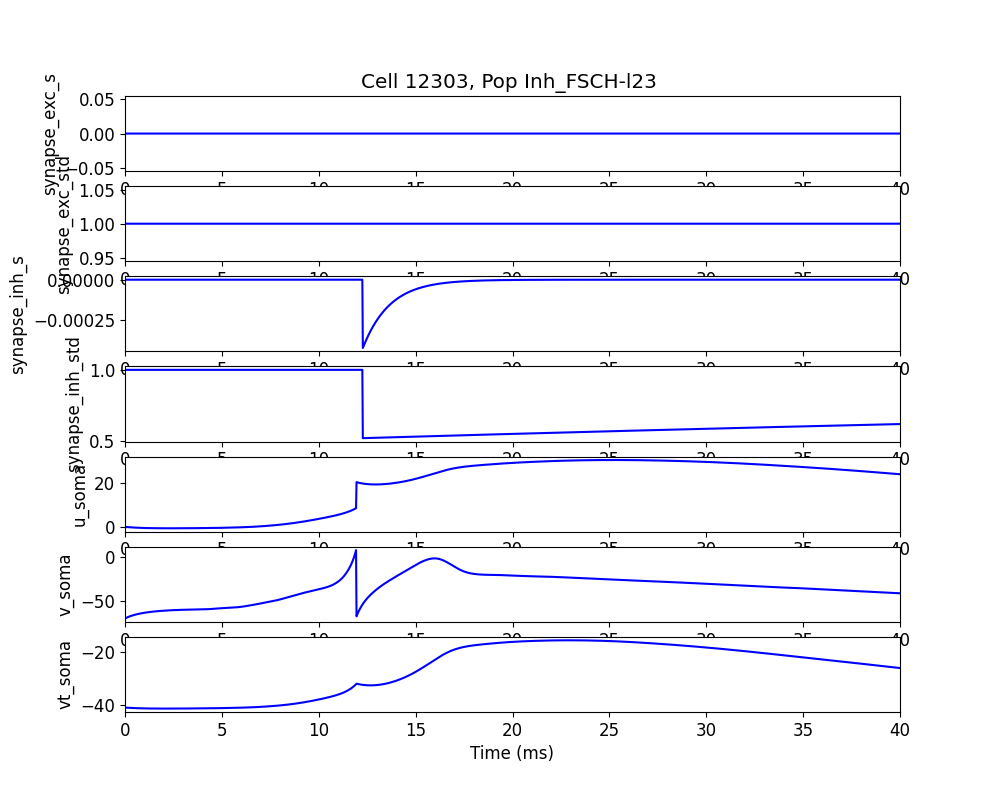
##### Layer 4 neurons

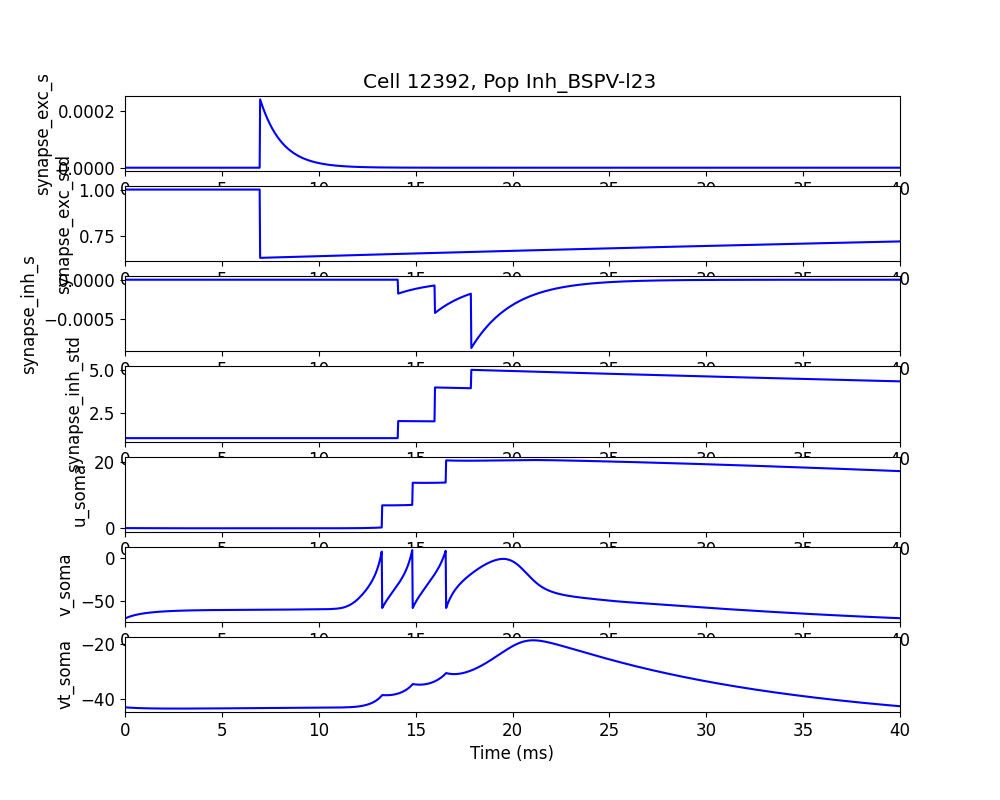


##### Layer 2/3 neurons (some pops)





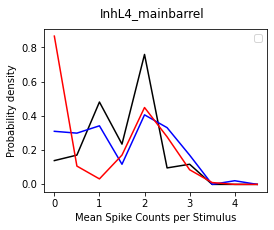
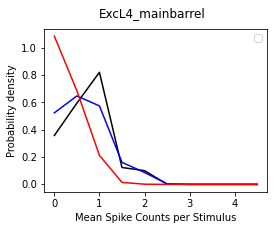


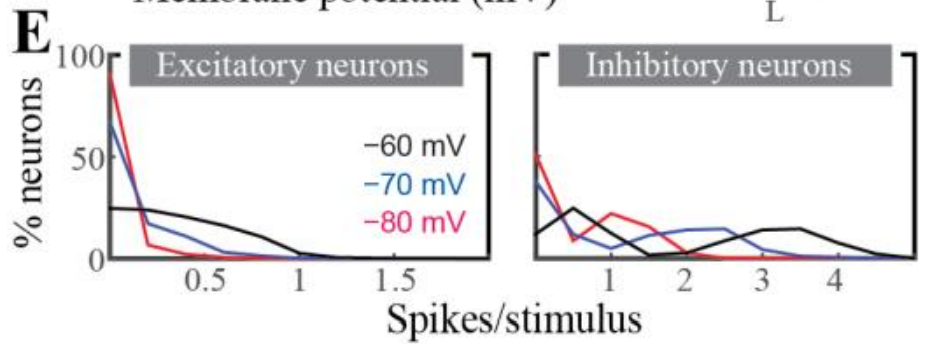


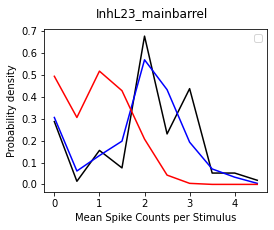
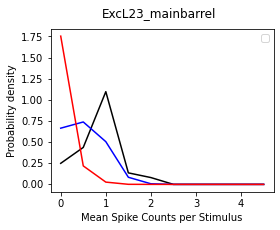
#### Spike counts per stimulus (stats)

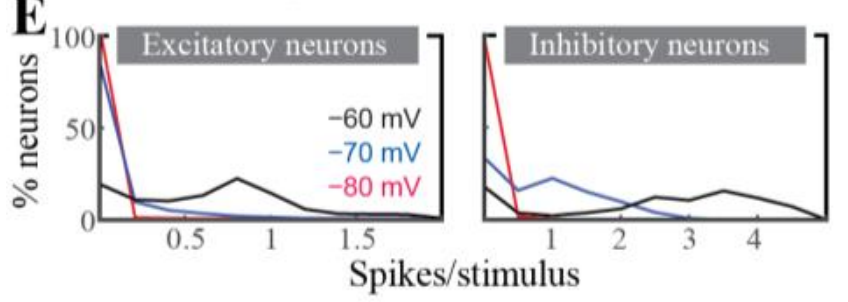
Populations grouped broadby by exc/inh and by layer. Compared with Paper figs 4E and 5E.

##### Main barrel

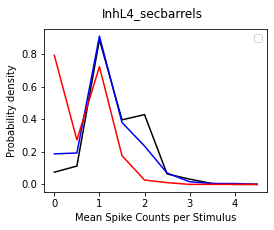
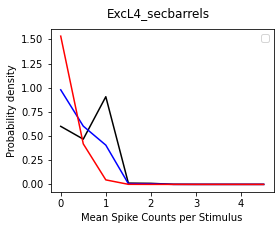


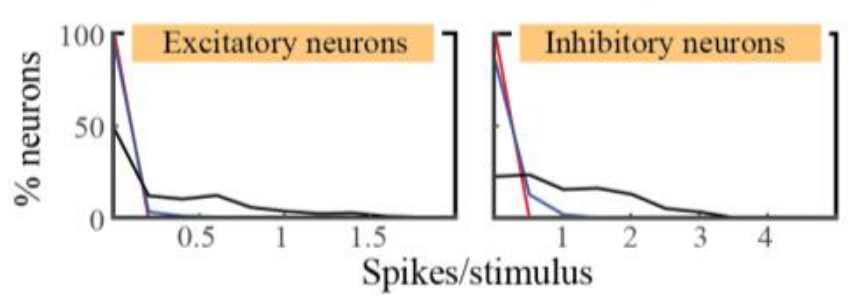


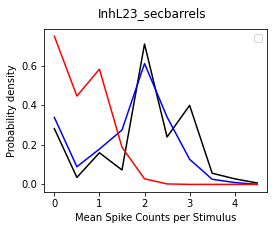
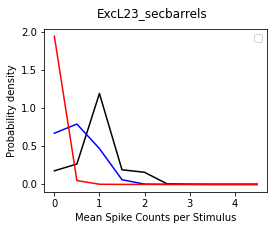


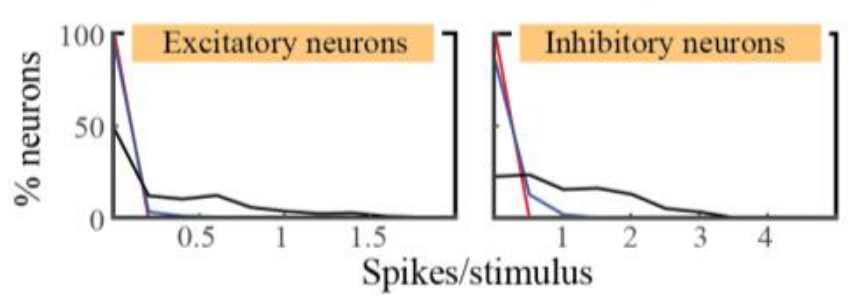


##### Secondary barrels



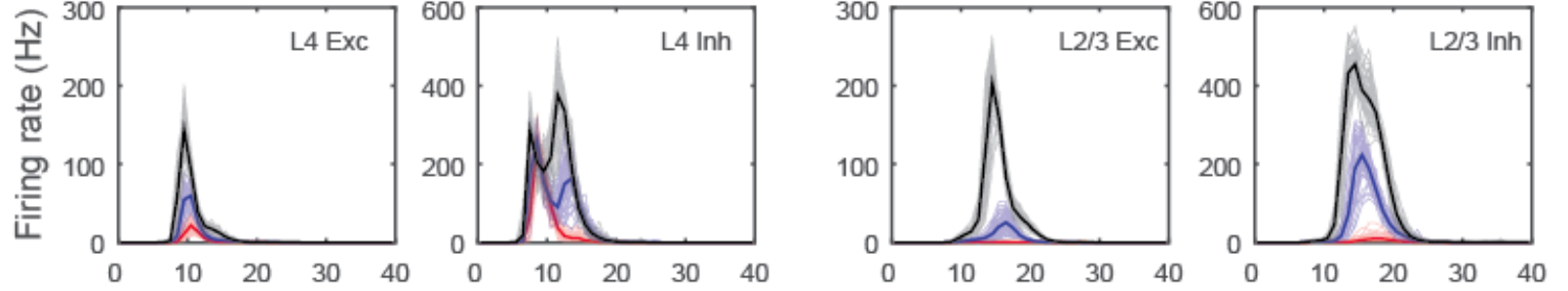


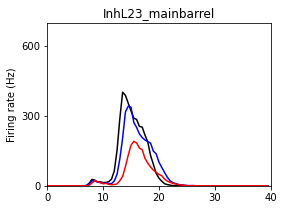
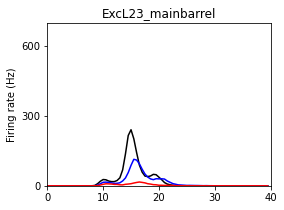
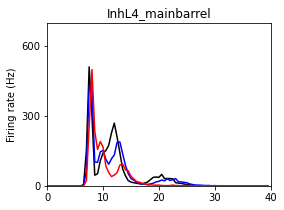
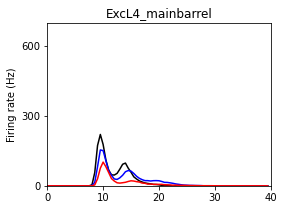




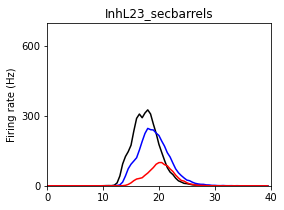
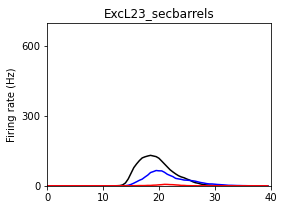
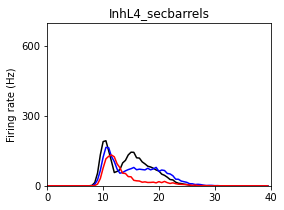
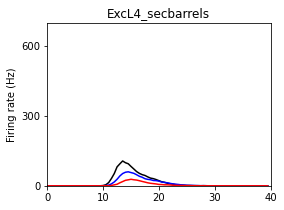
#### Peri-stimulus time histograms (PSTHs)

**Average across trials (mail barrel)**





Secondary barrels



There are some differences with the paper. Their origin may be from two sources:

1. There are some differences between the PSTHs used in these simulations and those used in the paper. This may have consequences mostly in the Psth of excitatory cells in L4 (here there is a second peak).
2. The adaptive threshold here is based on Fontaine et al., whereas the simulations in the paper corresponding to Figs. 2-6 where based on a different version of the adaptive threshold model. All simulations were repeated with a *constant* threshold (equal to the initial value). The overall spiking is about twice for vr = -60 mV, about half for vr = -70 mV, and a fraction for vr = -80mV. Statistics (mean spike counts per stimulus) in some cases are better represented by this case (for example, those of vr = -80mV), indicating that the previous adaptive threshold model used is somewhat in between the constant threshold and the Fontaine’s model.

In the folder “Analysis”, there are some scripts to run the above analysis (analyze.py). They are based on multitrial experiments in different “conditions”: vr = -60mV, vr = -70mV, vr = -80mV (set in analyze.py itself). Results from each of these conditions are put in folders “60mV”, “70mV”, “80mV”, respectively. Also, input folders used in the simulation are needed here to reconstruct some properties of the populations used to define the statistics (for example, which barrel a given id belongs to?).

1. https://www.neuron.yale.edu/neuron/static/py\_doc/modelspec/programmatic/topology/geometry.html [↑](#footnote-ref-1)