11- Advanced BMTK

Instructions

This document assumes you have completed the necessary steps in **02-Single_Cell_Hoc_BMTK** or **03-Networked_Hoc_BMTK**

Custom Cell Positions

Position lists and random positions. (from bmtk.builder.auxi.node_params import positions_list)

Custom Synapses

Included with the mod files from our HCO network from 03-Networked_Hoc_BMTK is a custom `inhsyn.mod` file. We can direct BMTK to use this file by taking the following steps

1. Copy the entire `my_bmtk_model` folder to a new folder called `my_bmtk_model_syn`.
2. In this new folder, create a new file named `synapses.py` and add the following code to this file:

synapses.py

```
 1  from bmtk.simulator.bionet.pyfunction_cache import add_synapse_model
 2  from neuron import h
 3
 4  def InhSyn(syn_params, sec_x, sec_id):
 5      """Create a inhsyn synapse
 6      :param syn_params: parameters of a synapse
 7      :param sec_x: normalized distance along the section
 8      :param sec_id: target section
 9      :return: NEURON synapse object
10      """
11
12      lsyn = h.inhsyn(sec_x, sec=sec_id)
13
14      if syn_params.get('esyn'):
15          lsyn.esyn = float(syn_params['esyn'])
16      if syn_params.get('gmax'):
17          lsyn.gmax = float(syn_params['gmax'])
18
19      return lsyn
20
21  def inhsyn(syn_params, xs, secs):
22      """Create a list of inhsyn synapses
23      :param syn_params: parameters of a synapse
24      :param xs: list of normalized distances along the section
25      :param secs: target sections
26      :return: list of NEURON synpase objects
27      """
28      syns = []
29      for x, sec in zip(xs, secs):
```

```
30          syn = InhSyn(syn_params, x, sec)
31          syns.append(syn)
32      return syns
33
34  def load():
34      add_synapse_model(InhSyn, 'inhsyn', overwrite=False)
35      add_synapse_model(InhSyn, overwrite=False)
        return
```

3. Things to note:
    a. (To use this in your model simply change everywhere InhSyn and inhsyn is defined to your synapse name, with variable name changes to line 14+)
    b. Line 1: `add_synapse_model` function is called to add custom synapses to BMTK's python function cache, allowing BMTK to "see" and use your synapse file
    c. Line 4: `syn_params` will be a dictionary containing parameters defined in the json file referenced when creating edges (shown later)
    d. Line 12: `h.inhsyn` will instantiate the `inhsyn` neuron hobject
    e. Line 14-17: Set properties of the synapse by:
        i. Checking to see if the parameter has been defined
        ii. Setting the synapse value to the supplied `syn_params` value
    f. Line 19: Create an additional function for BMTK to handle lists of synapses, we simply link it to our previous InhSyn function to prevent code duplication
    g. Line 34: Call `load()` in your `build_network.py` and `run_bionet.py` scripts early on to notify BMTK that you have custom synapses. (also shown later)

4. Create a new file called `my_inhsyn.json` in `./biophys_components/synaptic_models/` and place the following into it (Note how `esyn` and `gmax` appear in this file and the synapse function we defined previously)

my_inhsyn.json

```
1  {
2    "esyn":"-80",
3    "gmax":"40e-3"
4  }
5
```

5. In `build_network1.py`, we can now import this synapse file, call the load function, and use the synapse when defining our edges. See the complete file below with explanations.

build_network1.py

```
1  from bmtk.builder.networks import NetworkBuilder
2  import synapses
3
4  synapses.load()
5
6  net1 = NetworkBuilder('hco_net')
```

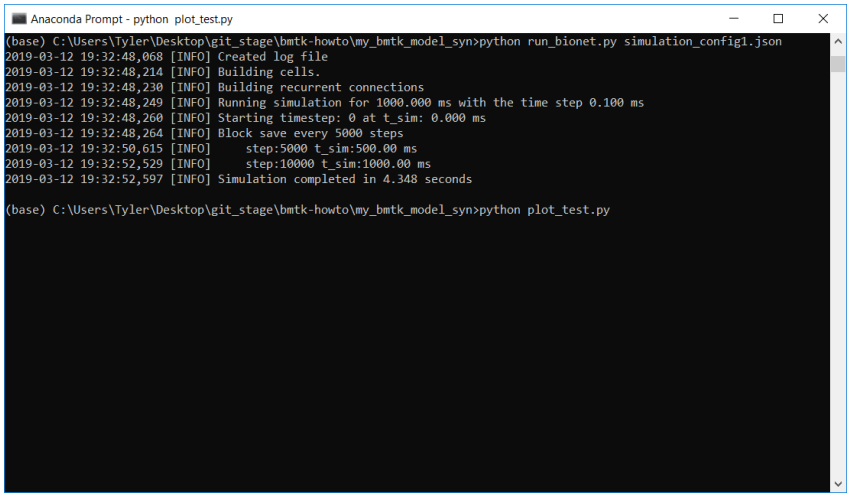
```
 7 net1.add_nodes(N=1,
 8                cell_name='HCOCell1',
 9                model_type='biophysical',
10                model_template='hoc:HCOcell',
11                morphology='blank.swc'
12              )
13
14 net1.add_nodes(N=1,
15                cell_name='HCOCell2',
16                model_type='biophysical',
17                model_template='hoc:HCOcell',
18                morphology='blank.swc'
19              )
20
21
22
23 net1.add_edges(source={'cell_name': 'HCOCell1'},
24 target={'cell_name':'HCOCell2'},
25                connection_rule=1,
26                syn_weight=40.0e-02,
27                dynamics_params='my_inhsyn.json',
28                model_template='inhsyn',
29                delay=0.0,
30                target_sections=["soma"],
31                distance_range=[0,999])
32
33 net1.add_edges(source={'cell_name': 'HCOCell2'},
34 target={'cell_name':'HCOCell1'},
34                connection_rule=1,
35                syn_weight=40.0e-02,
36                dynamics_params='my_inhsyn.json',
37                model_template='inhsyn',
38                delay=0.0,
39                target_sections=["soma"],
40                distance_range=[0,999])
41
42 net1.build()
43 net1.save_nodes(output_dir='network')
44
45 net1.build()
46 net1.save_edges(output_dir='network')
47
```

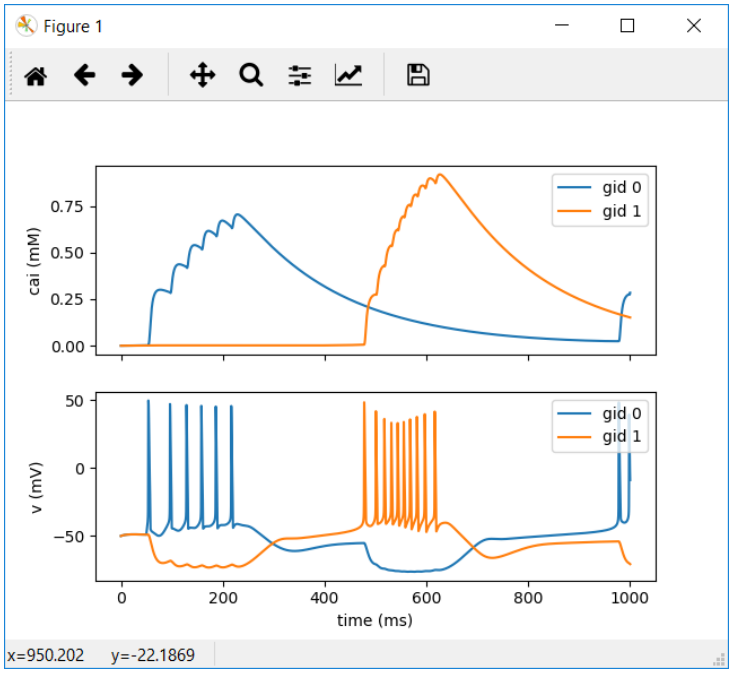
6. Things to note:
    a. Lines 2 and 4: import the synapses file we just created and load the synapses by calling the `load` function.
    b. Lines 27,28, 36,37: reference the synapse name and dynamics_params file json created earlier
7. Run `` `python build_network1.py` `` to build the network.
8. Add the synapse load function to the top of run_bionet.py like the following:

run_bionet.py (snippet)

```
1  import os, sys
2  from bmtk.simulator import bionet
3  from bmtk.simulator.bionet.default_setters.cell_models import loadHOC
4  import synapses
5
6  synapses.load()
7  bionet.pyfunction_cache.add_cell_model(loadHOC, directive='hoc',
8  model_type='biophysical')
9
```

9. You are now ready to run your network, run `python run_bionet.py simulation_config1.json` then `python plot_test.py`. You should notice small changes in the dynamics of the synapse output from previous tests.





Dynamic Synapse Properties

Edge properties like delay can be changed dynamically, per connection, rather than a blanket set value.

Recurrent Synapses

Connect neurons back to neurons they're connected to. (See https://gist.github.com/tjbanks/8228e341e33f65d641bccc6f187e0895 until instructions can be made)

Rule Based Synapses

If we want to define connections very specifically we can make use of the "all_to_one" iterator when adding edges.