

# 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 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:

	<b>synapses.py</b>
1	from bmtk.simulator.bionet.pyfunction_cache import <b>add_synapse_model</b>
2	from neuron import h
3	
4	def InhSyn( <b>syn_params</b> , 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 = <b>h.inhsyn</b> (sec_x, sec=sec_id)
13	
14	<b>if syn_params.get('esyn'):</b>
15	<b>lsyn.esyn = float(syn_params['esyn'])</b>
16	<b>if syn_params.get('gmax'):</b>
17	<b>lsyn.gmax = float(syn_params['gmax'])</b>
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 synapse 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 <b>load</b> ():
34	<b>add_synapse_model</b> (InhSyn, 'inhsyn', overwrite=False)

35	<code>add_synapse_model(InhSyn, overwrite=False)</code> <code>return</code>
----	--

### 3. Things to note:

- (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+)
- 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
- Line 4: `syn_params` will be a dictionary containing parameters defined in the json file referenced when creating edges (shown later)
- Line 12: `h.inhsyn` will instantiate the `inhsyn` neuron hobject
- Line 14-17: Set properties of the synapse by:
  - Checking to see if the parameter has been defined
  - Setting the synapse value to the supplied `syn_params` value
- 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
- 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)

<code>my_inhsyn.json</code>	
1	<code>{</code>
2	<code>  "esyn": "-80",</code>
3	<code>  "gmax": "40e-3"</code>
4	<code>}</code>
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.

<code>build_network1.py</code>	
1	<code>from bmtk.builder.networks import NetworkBuilder</code>
2	<code>import synapses</code>
3	
4	<code>synapses.load()</code>
5	
6	<code>net1 = NetworkBuilder('hco_net')</code>
7	<code>net1.add_nodes(N=1,</code>
8	<code>                cell_name='HCOCell11',</code>
9	<code>                model_type='biophysical',</code>
10	<code>                model_template='hoc:HCOcell',</code>
11	<code>                morphology='blank.swc'</code>
12	<code>)</code>

```

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'},
35                   connection_rule=1,
36                   syn_weight=40.0e-02,
37                   dynamics_params='my_inhsyn.json',
38                   model_template='inhsyn',
39                   delay=0.0,
40                   target_sections=["soma"],
41                   distance_range=[0,999])
42
43 net1.build()
44 net1.save_nodes(output_dir='network')
45
46 net1.build()
47 net1.save_edges(output_dir='network')

```

#### 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:

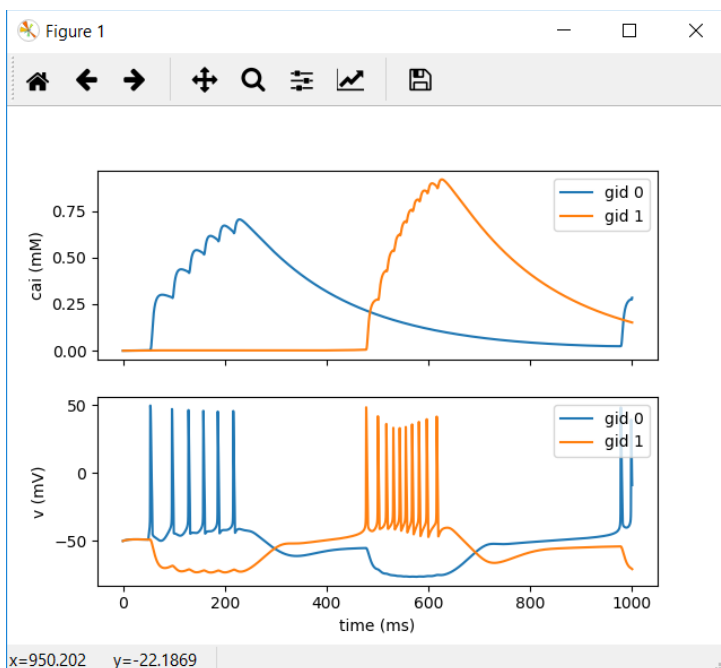
run_bionet.py (snippet)	
1	import os, sys
2	from bmtk.simulator import bionet
3	from bmtk.simulator.bionet.default_setters.cell_models import loadHOC
4	import synapses
5	
6	synapses.load()
7	bionet.pyfunction_cache.add_cell_model(loadHOC, directive='hoc',
8	model_type='biophysical')
9	

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.

```

Anaconda Prompt - python plot_test.py
(base) C:\Users\Tyler\Desktop\git_stage\bmtk-howto\my_bmtk_model_syn>python run_bionet.py simulation_config1.json
2019-03-12 19:32:48,068 [INFO] Created log file
2019-03-12 19:32:48,214 [INFO] Building cells.
2019-03-12 19:32:48,230 [INFO] Building recurrent connections
2019-03-12 19:32:48,249 [INFO] Running simulation for 1000,000 ms with the time step 0.100 ms
2019-03-12 19:32:48,260 [INFO] Starting timestep: 0 at t_sim: 0.000 ms
2019-03-12 19:32:48,264 [INFO] Block save every 5000 steps
2019-03-12 19:32:50,615 [INFO] step:5000 t_sim:500.00 ms
2019-03-12 19:32:52,529 [INFO] step:10000 t_sim:1000.00 ms
2019-03-12 19:32:52,597 [INFO] Simulation completed in 4.348 seconds
(base) C:\Users\Tyler\Desktop\git_stage\bmtk-howto\my_bmtk_model_syn>python plot_test.py

```



## Custom Cell Positions

**Columnar cell positioning** and cell **rotation** with **randomized rotations**:

[https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/03\\_single\\_pop.ipynb](https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/03_single_pop.ipynb)

## Dynamic Synapse Properties

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

See the following for a great example use case:

[https://github.com/AllenInstitute/bmtk/blob/develop/docs/examples/bio\\_450cells\\_exact/build\\_network.py](https://github.com/AllenInstitute/bmtk/blob/develop/docs/examples/bio_450cells_exact/build_network.py)

	snippet.py
1	def <b>build_edges</b> (src, trg, sections=['basal', 'apical'],
2	dist_range=[50.0, 150.0]):
3	"""Function used to randomly assign a synaptic location based on the
4	section (soma, basal, apical) and an
5	arc-length dist_range from the soma. This function should be passed
6	into the network and called during the build
7	process.
8	:param src: source cell (dict)
9	:param trg: target cell (dict)
10	:param sections: list of target cell sections to synapse onto
11	:param dist_range: range (distance from soma center) to place
12	:return:
13	"""
14	# Get morphology and soma center for the target cell
15	swc_reader = morphologies[trg['model_name']]
16	target_coords = [trg['x'], trg['y'], trg['z']]
17	
18	sec_ids, sec_xs = swc_reader.choose_sections(sections, dist_range)
19	# randomly choose sec_ids
20	coords = swc_reader.get_coord(sec_ids, sec_xs,
21	soma_center=target_coords) # get coords of sec_ids
22	dist = swc_reader.get_dist(sec_ids)
23	swctype = swc_reader.get_type(sec_ids)
24	return sec_ids, sec_xs, coords[0][0], coords[0][1], coords[0][2],
25	dist[0], swctype[0]
26	
27	...
28	
29	<b>cm</b> = <b>internal.add_edges</b> (source={'ei': 'e'}, target={'ei': 'e',
30	'model_type': 'biophysical'},
31	connection_rule=n_connections,
32	connection_params={'prob': 0.2},
33	dynamics_params='AMPA_ExcToExc.json',
34	model_template='Exp2Syn',
34	delay=2.0)
35	cm.add_properties('syn weight', rule=6.0e-05, dtypes=np.float)

```

36 cm.add_properties(['sec_id', 'sec_x', 'pos_x', 'pos_y', 'pos_z',
37 'dist', 'type'],
38                  rule=build_edges,
39                  rule_params={'sections': ['basal', 'apical'],
40 'dist_range': [30.0, 150.0]},
41 dtypes=[np.int32, np.float, np.float, np.float, np.float, np.float,
42 np.uint8])
43
44
45
46
47

```

## Recurrent Synapses

Example of connecting neurons back to neurons they're connected to. (Also at <https://gist.github.com/tjbanks/8228e341e33f65d641bccc6f187e0895>)

```

snippet.py
1 #####
2 # Build connections
3 #####
4 #Connect CA3o->CA3e Inhibitory
5 dynamics_file = 'CA3o2CA3e.inh.json'
6 conn = net.add_edges(source={'pop_name': 'CA3o'}, target={'pop_name':
7 'CA3e'},
8                    connection rule=hipp dist connector,
9
10 connection_params={'con_pattern':syn[dynamics_file]['con_pattern']},
11                    syn_weight=5.0e-03,
12                    dynamics_params=dynamics_file,
13                    model template=syn[dynamics file]['level of detail'],
14                    distance_range=[0.0, 300.0],
15                    target_sections=['soma'],
16                    delay=0.0)
17 conn.add_properties(['sec_id','sec_x'],rule=(0, 0.5),
18 dtypes=[np.int32,np.float])
19 conn.add_properties('delay',
20                    rule=syn_dist_delay,
21                    rule_params={'base_delay':syn[dynamics_file]['delay']},
22                    dtypes=np.float)
23
24 #####
25 # Build recurrent connection rules
26 #####
27 def hipp_recurrent_connector(source,target,all_edges=[],min_syn=1,
28 max_syn=1):
29     """
30     General logic:
31     1. Given a *potential* source and target
32     2. Look through all edges currently made

```

```

33     3. If any of the current edges contains
34         a. the current source as a previous target of
34         b. the current target as a previous source
35     4. Return number of synapses per this connection, 0 otherwise (no
36 connection)
37     """
38     for e in all_edges:
39         if source['node_id'] == e.target_gid and target['node_id'] ==
40 e.source_gid:
41         return random.randint(min_syn,max_syn)
42
43     return 0
44
45 #####
46 # Build recurrent connections
47 #####
48
49 #Connect CA3e->CA3o Excitatory
50 dynamics_file = 'CA3e2CA3o.exc.json'
51 conn = net.add_edges(source={'pop_name': 'CA3e'}, target={'pop_name':
52 'CA3o'},
53                     connection_rule=hipp_recurrent_connector,
54                     connection_params={'all_edges':net.edges()},
55                     syn_weight=5.0e-03,
56                     dynamics_params=dynamics_file,
57                     model_template=syn[dynamics_file]['level_of_detail'],
58                     distance_range=[0.0, 300.0],
59                     target_sections=['soma'],
60                     delay=0.0)
61 conn.add_properties(['sec_id','sec_x'],rule=(0, 0.5),
62 dtypes=[np.int32,np.float])
63 conn.add_properties('delay',
64                     rule=syn_dist_delay,
65
66 rule_params={'base_delay':syn[dynamics_file]['delay'],'dist_delay':0.1},
67 #Connect.hoc:274 0.1 dist delay
68 dtypes=np.float)
69

```

## Rule Based Synapses

If we want to define connections very specifically we can make use of the “all\_to\_one” iterator when adding edges.

	snippet.py
1	#####
2	# Build strict connection rules
3	#####
4	def hipp_MF_connector(source,targets,min_syn=1, max_syn=1):
5	"""
6	Exactly 2 connections from source to target
7	Pick a random in the target area
8	/docs/tutorial/04_multi_pop.ipynb:384:

```

9      "To tell the builder to use this schema, we must set
10 iterator='all_to_one'
11      in the add_edges method. (By default this is set to
12 'one_to_one'. You can
13      also use 'one_to_all' iterator which will pass in a single
14 source and all
15      possible targets)."
```

```

16      """
17
18      total_targets = len(targets)
19      syns = np.zeros(total_targets)
20      x_ind = 0
21      n = 0
22      while n < 2:
23          target_index = random.randint(0,total_targets-1)
24          target = targets[target_index]
25          dx = target['positions'][x_ind] - source['positions'][x_ind]
26
27          #prob = 1/ (exp( ((abs(dx) -0)^2)/ (2 * (2^2)))) //
28 Standard deviation of 2 compared to 3 in pp projections More limited
29 longitudinal spread
30          prob = 1/(math.exp(((abs(dx)-0)**2)/(2*(2**2))))
31          if random.random() < prob:
32              n=n+1
33              syns[target_index] = random.randint(min_syn,max_syn)
34
35      return syns
36
37 #####
38 # Build strict connections
39 #####
40
41 #Connect DGg->CA3e Excitatory (Exactly 2 connections allowed)
42 NOTICE: iterator is 'one_to_all'
43 dynamics_file = 'DGg2CA3e.exc.json'
44 conn = net.add_edges(source={'pop_name': 'DGg'}, target={'pop_name':
45 'CA3e'},
46                      iterator='one_to_all',
47                      connection_rule=hipp MF connector,
48                      connection_params={},
49                      syn_weight=5.0e-03,
50                      dynamics_params=dynamics_file,
51                      model_template=syn[dynamics_file]['level_of_detail'],
52                      distance_range=[0.0, 300.0],
53                      target_sections=['soma'],
54                      delay=0.0)
55 conn.add_properties(['sec_id','sec_x'],rule=(0, 0.5),
56 dtypes=[np.int32,np.float])
57 conn.add_properties('delay',
58                     rule=syn_dist_delay,
59                     rule_params={'base_delay':syn[dynamics_file]['delay']},
60 #Connect.hoc:274 0.1 dist delay
61 dtypes=np.float)
62

```