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:

|  |  |
| --- | --- |
|  | **synapses.py** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  34  35 | from bmtk.simulator.bionet.pyfunction\_cache import add\_synapse\_model  from neuron import h  def InhSyn(syn\_params, sec\_x, sec\_id):  """Create a inhsyn synapse  :param syn\_params: parameters of a synapse  :param sec\_x: normalized distance along the section  :param sec\_id: target section  :return: NEURON synapse object  """  lsyn = h.inhsyn(sec\_x, sec=sec\_id)  if syn\_params.get('esyn'):  lsyn.esyn = float(syn\_params['esyn'])  if syn\_params.get('gmax'):  lsyn.gmax = float(syn\_params['gmax'])    return lsyn  def inhsyn(syn\_params, xs, secs):  """Create a list of inhsyn synapses  :param syn\_params: parameters of a synapse  :param xs: list of normalized distances along the section  :param secs: target sections  :return: list of NEURON synpase objects  """  syns = []  for x, sec in zip(xs, secs):  syn = InhSyn(syn\_params, x, sec)  syns.append(syn)  return syns  def load():  add\_synapse\_model(InhSyn, 'inhsyn', overwrite=False)  add\_synapse\_model(InhSyn, overwrite=False)  return |

1. Things to note:
   1. (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+)
   2. 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
   3. Line 4: syn\_params will be a dictionary containing parameters defined in the json file referenced when creating edges (shown later)
   4. Line 12: h.inhsyn will instantiate the inhsyn neuron hobject
   5. Line 14-17: Set properties of the synapse by:
      1. Checking to see if the parameter has been defined
      2. Setting the synapse value to the supplied syn\_params value
   6. 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
   7. 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)
2. 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  3  4  5 | {  "esyn":"-80",  "gmax":"40e-3"  } |

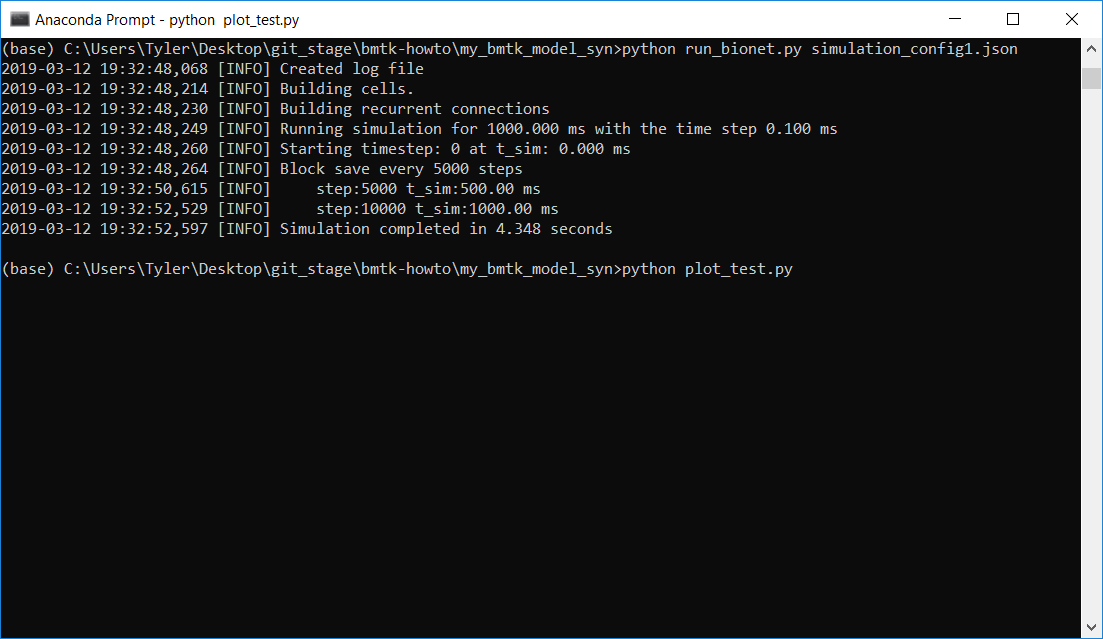
1. In build\_network1.py, we can now import this synapse file, call the load function, and use the synapse when defining our edges. See the complete file below with explanations.

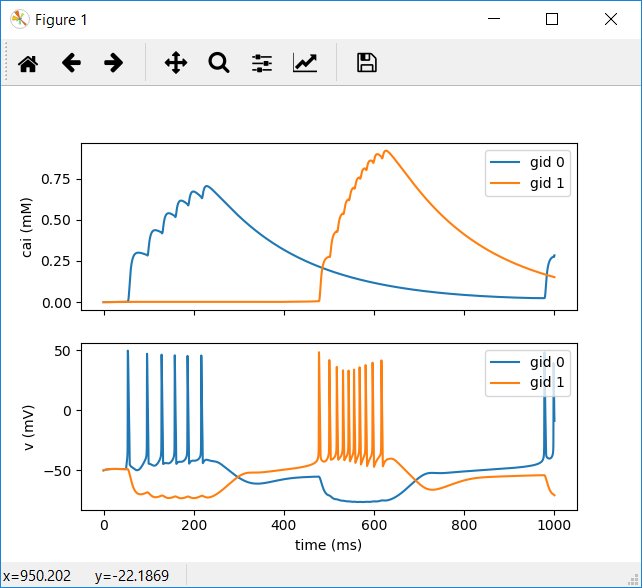
|  |  |
| --- | --- |
|  | **build\_network1.py** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | from bmtk.builder.networks import NetworkBuilder  import synapses  synapses.load()  net1 = NetworkBuilder('hco\_net')  net1.add\_nodes(N=1,  cell\_name='HCOCell1',  model\_type='biophysical',  model\_template='hoc:HCOcell',  morphology='blank.swc'  )    net1.add\_nodes(N=1,  cell\_name='HCOCell2',  model\_type='biophysical',  model\_template='hoc:HCOcell',  morphology='blank.swc'  )      net1.add\_edges(source={'cell\_name': 'HCOCell1'}, target={'cell\_name':'HCOCell2'},  connection\_rule=1,  syn\_weight=40.0e-02,  dynamics\_params='my\_inhsyn.json',  model\_template='inhsyn',  delay=0.0,  target\_sections=["soma"],  distance\_range=[0,999])    net1.add\_edges(source={'cell\_name': 'HCOCell2'}, target={'cell\_name':'HCOCell1'},  connection\_rule=1,  syn\_weight=40.0e-02,  dynamics\_params='my\_inhsyn.json',  model\_template='inhsyn',  delay=0.0,  target\_sections=["soma"],  distance\_range=[0,999])  net1.build()  net1.save\_nodes(output\_dir='network')    net1.build()  net1.save\_edges(output\_dir='network') |

1. Things to note:
   1. Lines 2 and 4: import the synapses file we just created and load the synapses by calling the load function.
   2. Lines 27,28, 36,37: reference the synapse name and dynamics\_params file json created earlier
2. Run `python build\_network1.py` to build the network.
3. Add the synapse load function to the top of run\_bionet.py like the following:

|  |  |
| --- | --- |
|  | **run\_bionet.py (snippet)** |
| 1  2  3  4  5  6  7  8  9 | import os, sys  from bmtk.simulator import bionet  from bmtk.simulator.bionet.default\_setters.cell\_models import loadHOC  import synapses  synapses.load()  bionet.pyfunction\_cache.add\_cell\_model(loadHOC, directive='hoc', model\_type='biophysical') |

1. 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.





# 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>

# 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>

|  |  |
| --- | --- |
|  | **snippet.py** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | def build\_edges(src, trg, sections=['basal', 'apical'], dist\_range=[50.0, 150.0]):  """Function used to randomly assign a synaptic location based on the section (soma, basal, apical) and an  arc-length dist\_range from the soma. This function should be passed into the network and called during the build  process.  :param src: source cell (dict)  :param trg: target cell (dict)  :param sections: list of target cell sections to synapse onto  :param dist\_range: range (distance from soma center) to place  :return:  """  # Get morphology and soma center for the target cell  swc\_reader = morphologies[trg['model\_name']]  target\_coords = [trg['x'], trg['y'], trg['z']]  sec\_ids, sec\_xs = swc\_reader.choose\_sections(sections, dist\_range) # randomly choose sec\_ids  coords = swc\_reader.get\_coord(sec\_ids, sec\_xs, soma\_center=target\_coords) # get coords of sec\_ids  dist = swc\_reader.get\_dist(sec\_ids)  swctype = swc\_reader.get\_type(sec\_ids)  return sec\_ids, sec\_xs, coords[0][0], coords[0][1], coords[0][2], dist[0], swctype[0]  …  cm = internal.add\_edges(source={'ei': 'e'}, target={'ei': 'e', 'model\_type': 'biophysical'},  connection\_rule=n\_connections,  connection\_params={'prob': 0.2},  dynamics\_params='AMPA\_ExcToExc.json',  model\_template='Exp2Syn',  delay=2.0)  cm.add\_properties('syn\_weight', rule=6.0e-05, dtypes=np.float)  cm.add\_properties(['sec\_id', 'sec\_x', 'pos\_x', 'pos\_y', 'pos\_z', 'dist', 'type'],  rule=build\_edges,  rule\_params={'sections': ['basal', 'apical'], 'dist\_range': [30.0, 150.0]},  dtypes=[np.int32, np.float, np.float, np.float, np.float, np.float, np.uint8]) |

# 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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | ###########################################################  # Build connections  ###########################################################  #Connect CA3o->CA3e Inhibitory  dynamics\_file = 'CA3o2CA3e.inh.json'  conn = net.add\_edges(source={'pop\_name': 'CA3o'}, target={'pop\_name': 'CA3e'},  connection\_rule=hipp\_dist\_connector,  connection\_params={'con\_pattern':syn[dynamics\_file]['con\_pattern']},  syn\_weight=5.0e-03,  dynamics\_params=dynamics\_file,  model\_template=syn[dynamics\_file]['level\_of\_detail'],  distance\_range=[0.0, 300.0],  target\_sections=['soma'],  delay=0.0)  conn.add\_properties(['sec\_id','sec\_x'],rule=(0, 0.5), dtypes=[np.int32,np.float])  conn.add\_properties('delay',  rule=syn\_dist\_delay,  rule\_params={'base\_delay':syn[dynamics\_file]['delay']},  dtypes=np.float)  ###########################################################  # Build recurrent connection rules  ###########################################################  def hipp\_recurrent\_connector(source,target,all\_edges=[],min\_syn=1, max\_syn=1):  """  General logic:  1. Given a \*potential\* source and target  2. Look through all edges currently made  3. If any of the current edges contains  a. the current source as a previous target of  b. the current target as a prevous source  4. Return number of synapses per this connection, 0 otherwise (no connection)  """  for e in all\_edges:  if source['node\_id'] == e.target\_gid and target['node\_id'] == e.source\_gid:  return random.randint(min\_syn,max\_syn)  return 0  ###########################################################  # Build recurrent connections  ###########################################################  #Connect CA3e->CA3o Excitatory  dynamics\_file = 'CA3e2CA3o.exc.json'  conn = net.add\_edges(source={'pop\_name': 'CA3e'}, target={'pop\_name': 'CA3o'},  connection\_rule=hipp\_recurrent\_connector,  connection\_params={'all\_edges':net.edges()},  syn\_weight=5.0e-03,  dynamics\_params=dynamics\_file,  model\_template=syn[dynamics\_file]['level\_of\_detail'],  distance\_range=[0.0, 300.0],  target\_sections=['soma'],  delay=0.0)  conn.add\_properties(['sec\_id','sec\_x'],rule=(0, 0.5), dtypes=[np.int32,np.float])  conn.add\_properties('delay',  rule=syn\_dist\_delay,  rule\_params={'base\_delay':syn[dynamics\_file]['delay'],'dist\_delay':0.1}, #Connect.hoc:274 0.1 dist delay  dtypes=np.float) |

# 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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | ###########################################################  # Build strict connection rules  ###########################################################  def hipp\_MF\_connector(source,targets,min\_syn=1, max\_syn=1):  """  Exactly 2 connections from source to target  Pick a random in the target area  /docs/tutorial/04\_multi\_pop.ipynb:384:  "To tell the builder to use this schema, we must set iterator='all\_to\_one'  in the add\_edges method. (By default this is set to 'one\_to\_one'. You can  also use 'one\_to\_all' iterator which will pass in a single source and all  possible targets)."  """    total\_targets = len(targets)  syns = np.zeros(total\_targets)  x\_ind = 0  n = 0  while n < 2:  target\_index = random.randint(0,total\_targets-1)  target = targets[target\_index]  dx = target['positions'][x\_ind] - source['positions'][x\_ind]  #prob = 1/ (exp( ((abs(dx) -0)^2)/ (2 \* (2^2)))) // Standard deviation of 2 compared to 3 in pp projections More limited longitudianal spread  prob = 1/(math.exp(((abs(dx)-0)\*\*2)/(2\*(2\*\*2))))  if random.random() < prob:  n=n+1  syns[target\_index] = random.randint(min\_syn,max\_syn)  return syns  ###########################################################  # Build strict connections  ###########################################################  #Connect DGg->CA3e Excitatory (Exactly 2 connections allowed) NOTICE: iterator is 'one\_to\_all'  dynamics\_file = 'DGg2CA3e.exc.json'  conn = net.add\_edges(source={'pop\_name': 'DGg'}, target={'pop\_name': 'CA3e'},  iterator='one\_to\_all',  connection\_rule=hipp\_MF\_connector,  connection\_params={},  syn\_weight=5.0e-03,  dynamics\_params=dynamics\_file,  model\_template=syn[dynamics\_file]['level\_of\_detail'],  distance\_range=[0.0, 300.0],  target\_sections=['soma'],  delay=0.0)  conn.add\_properties(['sec\_id','sec\_x'],rule=(0, 0.5), dtypes=[np.int32,np.float])  conn.add\_properties('delay',  rule=syn\_dist\_delay,  rule\_params={'base\_delay':syn[dynamics\_file]['delay']}, #Connect.hoc:274 0.1 dist delay  dtypes=np.float) |