ConnPlotter: A Tutorial

Author: Hans Ekkehard Plesser
Institution: Norwegian University of Life Sciences, Simula Research Laboratory, RIKEN Brain Sciences Institute
Version: 0.7
Date: 1 December 2009

Introduction

This tutorial gives a brief introduction to the ConnPlotter toolbox. It is by no means complete.

Avoid interactive backend

```
20 import matplotlib
21 matplotlib.use("Agg")
```

Import pylab to call pylab.show() so that pyreport can capture figures created. Must come before import ConnPlotter so we get the correct show().

```
26 import pylab
```

Import ConnPlotter and its examples

```
29 import ConnPlotter as cpl
```

```
ConnPlotter Copyright (C) 2009 Hans Ekkehard Plesser/UMB

ConnPlotter comes with ABSOLUTELY NO WARRANTY.
ConnPlotter is free software, and you are welcome to redistribute it
under certain conditions. See GNU Public License v.3 or later for details.
```

```
30 import ConnPlotter.examples as ex
```

Turn of warnings about resized figure windows

```
33 import warnings
34 warnings.simplefilter("ignore")
```

Define a helper function to show LaTeX tables on the fly

```
48 def showTextTable(connPattern, fileTrunk):
49     """
50     Shows a Table of Connectivity as textual table.
51
52     Arguments:
53     connPattern  ConnectionPattern instance
54     fileTrunk    Eventual PNG image will be fileTrunk.png
55     """
56     import subprocess as subp # to call LaTeX etc
57
58     import os                 # to remove files
59
60     # Write to LaTeX file so we get a nice textual representation
61     # We want a complete LaTeX document, so we set ``standalone``
62     # to ``True``.
63     connPattern.toLaTeX(file=fileTrunk+'.tex', standalone=True, enumerate=
           True)
64     # Create PDF, crop, and convert to PNG
65     try:
66         devnull = open('/dev/null', 'w')
67         subp.call(['pdflatex', fileTrunk], stdout=devnull, stderr=subp.
               STDOUT)
```

```
68          # need wrapper, since pdfcrop does not begin with #!
69          subp.call(['pdfcrop_wrapper.sh', fileTrunk+'.pdf'],
70                    stdout=devnull, stderr=subp.STDOUT)
71          devnull.close()
72          os.rename(fileTrunk+'-crop.pdf', fileTrunk+'.pdf')
73          for suffix in ('.tex', '-crop.pdf', '.png', '.aux', '.log'):
74              if os.path.exists(fileTrunk + suffix):
75                  os.remove(fileTrunk + suffix)
76      except:
77          raise Exception('Could not create PDF Table.')
```

Simple network

This is a simple network with two layers A and B; layer B has two populations, E and I. On the NEST side, we use only synapse type `static_synapse`. ConnPlotter then infers that synapses with positive weights should have type `exc`, those with negative weight type `inh`. Those two types are know to ConnPlotter.

Obtain layer, connection and model list from the example set

```
75 s_layer, s_conn, s_model = ex.simple()
```

Create Connection Pattern representation

```
78 s_cp = cpl.ConnectionPattern(s_layer, s_conn)
```

Show pattern as textual table (we cheat a little and include PDF directly)

```
81 showTextTable(s_cp, 'simple_tt')
```

Connectivity						
	Src	**Tgt**	**Syn**	**Wght**	**Mask**	**Kernel**
1	IG	RG/E	exc	2	≤ 0.2	0.8
2	IG	RG/I	exc	2	≤ 0.3	0.4
3	RG/E	RG/E	exc	2	$[(-0.4,-0.2),(+0.4,+0.2)]$	1
4	RG/E	RG/E	exc	2	$[(-0.2,-0.4),(+0.2,+0.4)]$	1
5	RG/E	RG/I	exc	5	≤ 0.5	$\mathcal{G}(p_0 = 1, \sigma = 0.1)$
6	RG/I	RG/E	inh	-3	≤ 0.25	$\mathcal{G}(p_0 = 1, \sigma = 0.2)$
7	RG/I	RG/I	inh	-0.5	≤ 1	0.5

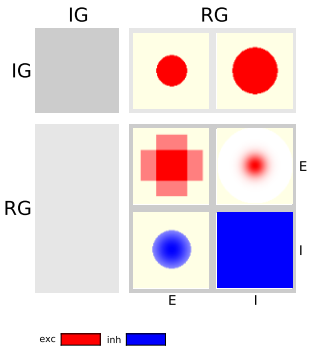
$\mathcal{G}(p_0, \sigma)$: $p(\mathbf{x}) = p_0 e^{-\mathbf{x}^2/2\sigma^2}$

Show pattern in full detail

A separate patch is shown for each pair of populations.

- Rows represent senders, columns targets.
- Layer names are given to the left/above, population names to the right and below.
- Excitatory synapses shown in blue, inhibitory in red.
- Each patch has its own color scale.

```
93 s_cp.plot()
94 pylab.show()
```

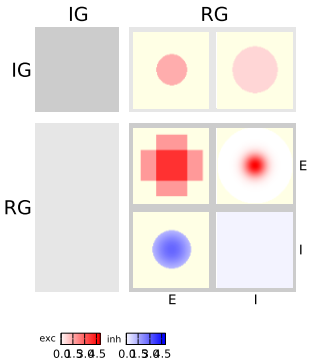


Let us take a look at what this connection pattern table shows:

- The left column, with header "A", is empty: The "A" layer receives no input.
- The right column shows input to layer "B"
  - The top row, labeled "A", has two patches in the "B" column:
    - The left patch shows relatively focused input to the "E" population in layer "B" (first row of "Connectivity" table).
    - The right patch shows wider input to the "I" population in layer "B" (second row of "Connectivity" table).
    - Patches are red, indicating excitatory connections.
    - In both cases, mask are circular, and the product of connection weight and probability is independent of the distance between sender and target neuron.
  - The grey rectangle to the bottom right shows all connections from layer "B" populations to layer "B" populations. It is subdivided into two rows and two columns:
    - Left column: inputs to the "E" population.
    - Right column: inputs to the "I" population.
    - Top row: projections from the "E" population.
    - Bottom row: projections from the "I" population.
    - There is only one type of synapse for each sender-target pair, so there is only a single patch per pair.
    - Patches in the top row, from population "E" show excitatory connections, thus they are red.
    - Patches in the bottom row, from population "I" show inhibitory connections, thus they are blue.
    - The patches in detail are:
      - **E to E** (top-left, row 3+4 in table): two rectangular projections at 90 degrees.
      - **E to I** (top-right, row 5 in table): narrow gaussian projection.
      - **I to E** (bottom-left, row 6 in table): wider gaussian projection.
      - **I to I** (bottom-right, row 7 in table): circular projection covering entire layer.
- **NB:** Color scales are different, so one **cannot** compare connection strengths between patches.

Full detail, common color scale

```
128 s_cp.plot(globalColors=True)
129 pylab.show()
```



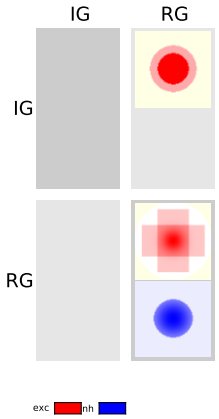
This figure shows the same data as the one above, but now all patches use a common color scale, so full intensity color (either red or blue) indicates the strongest connectivity. From this we see that

- A to B/E is stronger than A to B/I
- B/E to B/I is the strongest of all connections at the center
- B/I to B/E is stronger than B/I to B/I

Aggregate by groups

For each pair of population groups, sum connections of the same type across populations.

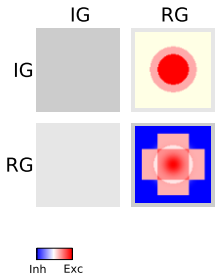
```
142 s_cp.plot(aggrGroups=True)
143 pylab.show()
```



In the figure above, all excitatory connections from B to B layer have been combined into one patch, as have all inhibitory connections from B to B. In the upper-right corner, all connections from layer A to layer B have been combined; the patch for inhibitory connections is missing, as there are none.

Aggregate by groups and synapse models

```
152 s_cp.plot(aggrGroups=True, aggrSyns=True)
153 pylab.show()
```

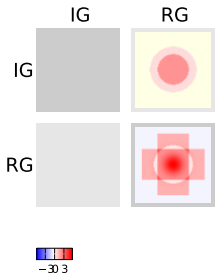


When aggregating across synapse models, excitatory and inhibitory connections are combined. By default, excitatory connections are weights with +1, inhibitory connections with -1 in the sum. This may yield kernels with positive and negative values. They are shown on a red-white-blue scale as follows:

- White always represents 0.
- Positive values are represented by increasingly saturated red.
- Negative values are represented by increasingly saturated blue.
- Colorscales are separate for red and blue:
  - largest positive value: fully saturated red
  - largest negative value: fully saturated blue
- Each patch has its own colorscales.

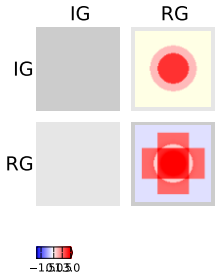
- When `aggrSyns=True` is combined with `globalColors=True`, all patches use the same minimum and maximum in their red and blue color scales. The the minimum is the negative of the maximum, so that blue and red intesities can be compared.

```
169 s_cp.plot(aggrGroups=True, aggrSyns=True, globalColors=True)
170 pylab.show()
```



- We can explicitly set the limits of the color scale; if values exceeding the limits are present, this is indicated by an arrowhead at the end of the colorbar. User-defined color limits need not be symmetric about 0.

```
173 s_cp.plot(aggrGroups=True, aggrSyns=True, globalColors=True, colorLimits=[
        -2,3])
174 pylab.show()
```



Save pattern to file

```
178 s_cp.plot(file='simple_example.png')
```

This saves the detailed diagram to the given file. If you want to save the pattern in several file formats, you can pass a tuple of file names, e.g., `s_cp.plot(file=('a.eps', 'a.png'))`.

NB: Saving directly to PDF may lead to files with artifacts. We recommend to save to EPS and the convert to PDF.

Build network in NEST

```
190 import nest
191 import nest.topology as topo
```

Create models

```
195 for model in s_model:
196     nest.CopyModel(model[0], model[1], model[2])
```

Create layers, store layer info in Python variable

```
199 for layer in s_layer:
200     exec '%s = topo.CreateLayer(layer[1])' % layer[0]
```

Create connections, need to insert variable names

```
203 for conn in s_conn:
204     eval('topo.ConnectLayer(%s,%s,conn[2])' % (conn[0], conn[1]))
205
206 nest.Simulate(10)
```

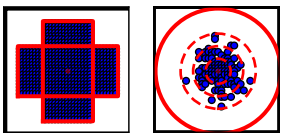
Ooops:* Nothing happened? Well, it did, but pyreport cannot capture the output directly generated by NEST. The absence of an error message in this place shows that network construction and simulation went through.

Inspecting the connections actually created

The following block of messy and makeshift code plots the targets of the center neuron of the B/E population in the B/E and the B/I populations.

```
214 B_top = nest.GetStatus(RG, 'topology')[0]
215 ctr_id = topo.GetElement(RG, [B_top['rows']/2, B_top['columns']/2])
216
217 # find excitatory element in B
218 E_id = [gid for gid in nest.GetLeaves(ctr_id)[0]
219         if nest.GetStatus([gid], 'model')[0] == 'E']
220
221 # get all targets, split into excitatory and inhibitory
222 alltgts = nest.GetStatus(nest.FindConnections(E_id, synapse_type='
        static_synapse'), 'target')
223 Etgts = [t for t in alltgts if nest.GetStatus([t], 'model')[0] == 'E']
224 Itgts = [t for t in alltgts if nest.GetStatus([t], 'model')[0] == 'I']
225
226 # obtain positions of targets
227 Etpos = zip(*[topo.GetPosition([n]) for n in Etgts])
228 Itpos = zip(*[topo.GetPosition([n]) for n in Itgts])
229
230 # plot excitatory
231 pylab.clf()
232 pylab.subplot(121)
233 pylab.scatter(Etpos[0], Etpos[1])
234 ctrpos = pylab.array(topo.GetPosition(E_id))
235 ax = pylab.gca()
236 ax.add_patch(pylab.Circle(ctrpos, radius=0.02, zorder = 99,
237                           fc = 'r', alpha = 0.4, ec = 'none'))
238 ax.add_patch(pylab.Rectangle(ctrpos + pylab.array((-0.4,-0.2)), 0.8, 0.4,
        zorder = 1,
239                              fc = 'none', ec = 'r', lw=3))
240 ax.add_patch(pylab.Rectangle(ctrpos + pylab.array((-0.2,-0.4)), 0.4, 0.8,
        zorder = 1,
241                              fc = 'none', ec = 'r', lw=3))
242 ax.add_patch(pylab.Rectangle(ctrpos + pylab.array((-0.5,-0.5)), 1.0, 1.0,
        zorder = 1,
243                              fc = 'none', ec = 'k', lw=3))
244 ax.set(aspect='equal', xlim=[-0.5,0.5], ylim=[-0.5,0.5],
245        xticks=[],yticks=[])
246
247 # plot inhibitory
248 pylab.subplot(122)
249 pylab.scatter(Itpos[0], Itpos[1])
250 ctrpos = topo.GetPosition(E_id)
251 ax = pylab.gca()
252 ax.add_patch(pylab.Circle(ctrpos, radius=0.02, zorder = 99,
253                           fc = 'r', alpha = 0.4, ec = 'none'))
254 ax.add_patch(pylab.Circle(ctrpos, radius=0.1, zorder = 2,
255                           fc = 'none', ec = 'r', lw=2, ls='dashed'))
256 ax.add_patch(pylab.Circle(ctrpos, radius=0.2, zorder = 2,
257                           fc = 'none', ec = 'r', lw=2, ls='dashed'))
258 ax.add_patch(pylab.Circle(ctrpos, radius=0.3, zorder = 2,
259                           fc = 'none', ec = 'r', lw=2, ls='dashed'))
260 ax.add_patch(pylab.Circle(ctrpos, radius=0.5, zorder = 2,
261                           fc = 'none', ec = 'r', lw=3))
```

```
ax.add_patch(pylab.Rectangle((-0.5,-0.5), 1.0, 1.0, zorder = 1,
                             fc = 'none', ec = 'k', lw=3))
ax.set(aspect='equal', xlim=[-0.5,0.5], ylim=[-0.5,0.5],
       xticks=[],yticks=[])
pylab.show()
```



Thick red lines mark the mask, dashed red lines to the right one, two and three standard deviations. The sender location is marked by the red spot in the center. Layers are 40x40 in size.

A more complex network

This network has layers A and B, with E and I populations in B. The added complexity comes from the fact that we now have four synapse types: AMPA, NMDA, GABA_A and GABA_B. These synapse types are known to ConnPlotter.

Setup and tabular display

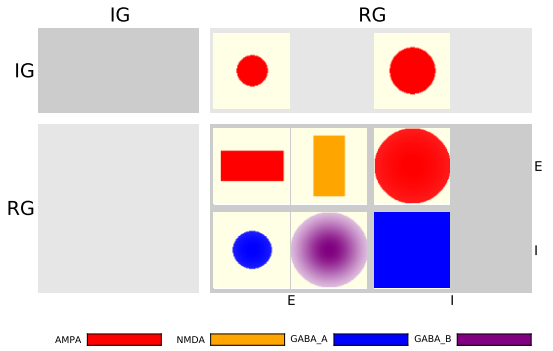
```
c_layer, c_conn, c_model = ex.complex()
c_cp = cpl.ConnectionPattern(c_layer, c_conn)
showTextTable(c_cp, 'complex_tt')
```

Connectivity						
	Src	**Tgt**	**Syn**	**Wght**	**Mask**	**Kernel**
1	IG	RG/E	AMPA	5	≤ 0.2	0.8
2	IG	RG/I	AMPA	2	≤ 0.3	0.4
3	RG/E	RG/E	AMPA	2	$[(-0.4,-0.2),(+0.4,+0.2)]$	1
4	RG/E	RG/E	NMDA	2	$[(-0.2,-0.4),(+0.2,+0.4)]$	1
5	RG/E	RG/I	AMPA	1	≤ 0.5	$\mathcal{G}(p_0=1,\sigma=1)$
6	RG/I	RG/E	GABA_A	-3	≤ 0.25	$\mathcal{G}(p_0=1,\sigma=0.5)$
7	RG/I	RG/E	GABA_B	-1	≤ 0.5	$\mathcal{G}(p_0=0.5,\sigma=0.3)$
8	RG/I	RG/I	GABA_A	-0.5	≤ 1	0.1

$\mathcal{G}(p_0,\sigma)$: $p(\mathbf{x}) = p_0 e^{-\mathbf{x}^2/2\sigma^2}$

Pattern in full detail

```
c_cp.plot()
pylab.show()
```



Note the following differences to the simple pattern case:

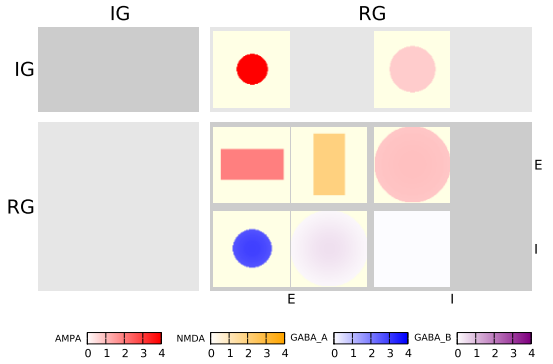
- For each pair of populations, e.g., B/E as sender and B/E as target, we now have two patches representing AMPA and NMDA synapse for the E population, GABA_A and _B for the I population.
- Colors are as follows:

  AMPA: red
  NMDA: orange
  GABA_A: blue
  GABA_B: purple

- Note that the horizontal rectangular pattern (table line 3) describes AMPA synapses, while the vertical rectangular pattern (table line 4) describes NMDA synapses.

Full detail, common color scale

```
302 c_cp.plot(globalColors=True)
303 pylab.show()
```

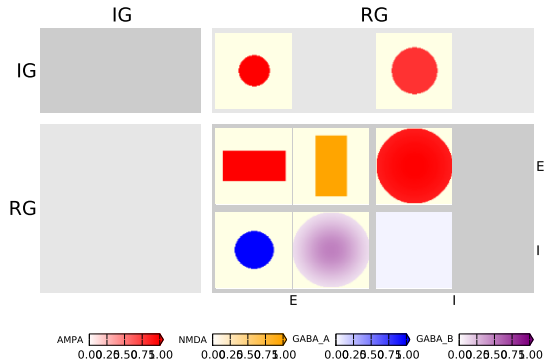


As above, but now with a common color scale. **NB:** The patch for the B/I to B/I connection may look empty, but it actually shows a very light shade of red. Rules are as follows:

- If there is no connection between two populations, show the grey layer background.
- All parts of the target layer that are outside the mask or strictly zero are off-white.
- If it looks bright white, it is a very diluted shade of the color for the pertaining synpase type.

Full detail, explicit color limits

```
314 c_cp.plot(colorLimits=[0,1])
315 pylab.show()
```



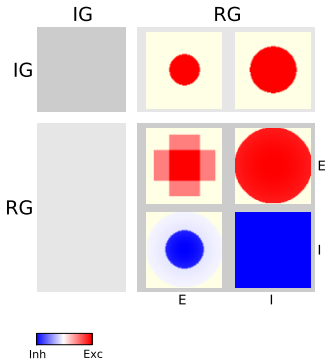
As above, but the common color scale is now given explicitly. The arrow at the right end of the color scale indicates that the values in the kernels extend beyond +1.

Aggregate by synapse models

For each population pair, connections are summed across synapse models.

- Excitatory kernels are weighted with +1, inhibitory kernels with -1.
- The resulting kernels are shown on a color scale ranging from red (inhibitory) via white (zero) to blue (excitatory).
- Each patch has its own color scale

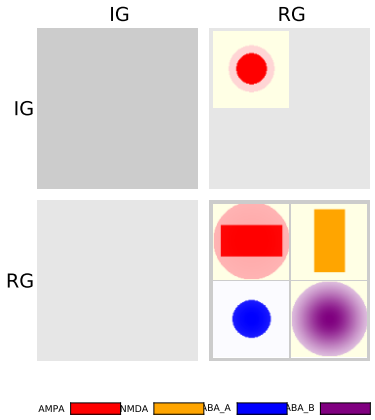
```
331 c_cp.plot(aggrSyns=True)
332 pylab.show()
```



- AMPA and NMDA connections from B/E to B/E are now combined to form a cross.
- GABA_A and GABA_B connections from B/I to B/E are two concentric spots.

Aggregate by population group

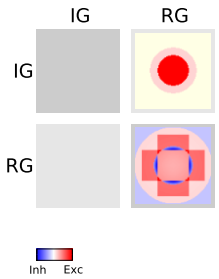
```
339 c_cp.plot(aggrGroups=True)
340 pylab.show()
```



This is in many ways orthogonal to aggregation by synapse model: We keep synapse types separat, while we combine across populations. Thus, we have added the horizonal bar (B/E to B/E, row 3) with the spot (B/E to B/I, row 5).

Aggregate by population group and synapse model

```
c_cp.plot(aggrGroups=True,aggrSyns=True)
pylab.show()
```



All connection are combined for each pair of sender/target layer.

CPTs using the total charge deposited (TCD) as intensity

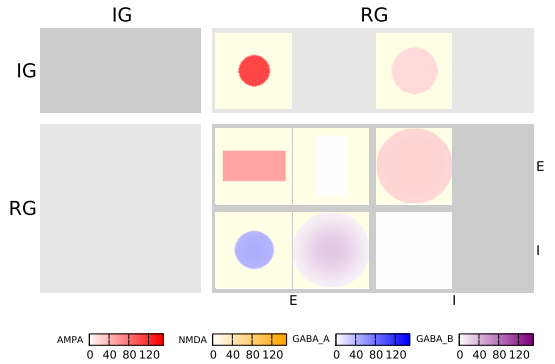
TCD-based CPTs are currently only available for the ht_neuron, since ConnPlotter does not know how to obtain int g(t) dt from NEST for other conductance-based model neurons. We need to create a separate ConnectionPattern instance for each membrane potential we want to use in the TCD computation

```
c_cp_75 = cpl.ConnectionPattern(c_layer, c_conn, intensity='tcd',
                                mList=c_model, Vmem=-75.0)
c_cp_45 = cpl.ConnectionPattern(c_layer, c_conn, intensity='tcd',
                                mList=c_model, Vmem=-45.0)
```

In order to obtain a meaningful comparison between both membrane potentials, we use the same global color scale

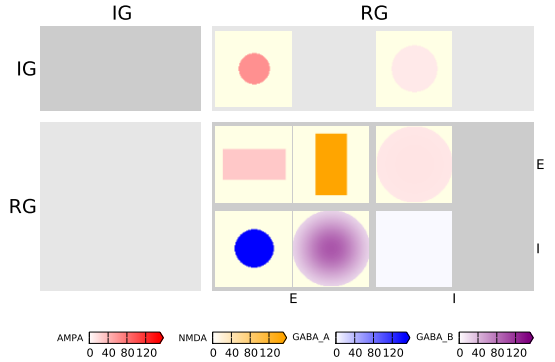
V_m = -75 mV

```
c_cp_75.plot(colorLimits=[0,150])
pylab.show()
```



V_m = -45 mV

```
374 c_cp_45.plot(colorLimits=[0,150])
375 pylab.show()
```



Note that the NMDA projection virtually vanishes for V_m=-75mV, but is very strong for V_m=-45mV. GABA_A and GABA_B projections are also stronger, while AMPA is weaker for V_m=-45mV.

Non-Dale network model

By default, ConnPlotter assumes that networks follow Dale's law, i.e., either make excitatory or inhibitory connections. If this assumption is violated, we need to inform ConnPlotter how synapse types are grouped. We look at a simple example here.

Load model

```
388 nd_layer, nd_conn, nd_model = ex.non_dale()
```

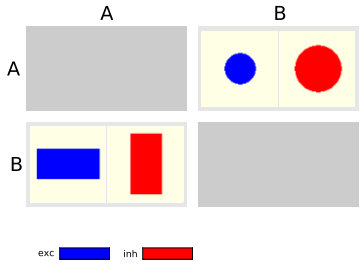
We specify the synapse configuration using the synTypes argument:

- synTypes is a tuple.
- Each element in the tuple represents a group of synapse models
- Any sender can make connections with synapses from **one group only**.
- Each synapse model is specified by a `SynType`.
- The SynType constructor takes three arguments:
  - The synapse model name
  - The weight to apply then aggregating across synapse models
  - The color to use for the synapse type
- Synapse names must be unique, and must form a superset of all synapse models in the network.

```
404 nd_cp = cpl.ConnectionPattern(nd_layer, nd_conn, synTypes=(
405             (cpl.SynType('exc', 1.0, 'b'), cpl.SynType('inh', -1.0, 'r')),)
                )
406 showTextTable(nd_cp, 'non_dale_tt')
```

Connectivity						
	Src	Tgt	Syn	Wght	Mask	Kernel
1	A	B	exc	2	≤ 0.2	0.8
2	A	B	inh	-2	≤ 0.3	0.4
3	B	A	exc	2	$[(-0.4,-0.2),(+0.4,+0.2)]$	1
4	B	A	inh	-2	$[(-0.2,-0.4),(+0.2,+0.4)]$	1

```
nd_cp.plot()
pylab.show()
```



Note that we now have red and blue patches side by side, as the same population can make excitatory and inhibitory connections.

Configuring the ConnectionPattern display

I will now show you a few ways in which you can configure how ConnPlotter shows connection patterns.

User defined synapse types

By default, ConnPlotter knows two following sets of synapse types.

exc/inh

- Used automatically when all connections have the same `synapse_model`.
- Connections with positive weight are assigned model exc, those with negative weight model inh.
- When computing totals, exc has weight +1, inh weight -1
- Exc is colored blue, inh red.

AMPA/NMDA/GABA_A/GABA_B

- Used if the set of `synapse_model` s in the network is a subset of those four types.
- AMPA/NMDA carry weight +1, GABA_A/GABA_B weight -1.
- Colors are as follows:

  AMPA: blue
  NMDA: green
  GABA_A: red
  GABA_B: magenta

We saw a first example of user-defined synapse types in the non-Dale example above. In that case, we only changed the grouping. Here, I will demonstrate the effect of different ordering, weighting, and color specifications. We use the complex model from above as example.

NOTE: It is most likey a *bad idea* to change the colors or placement of synapse types. If everyone uses the same design rules, we will all be able to read each others figures much more easily.

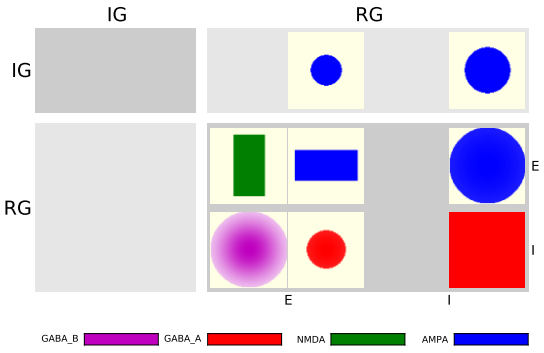
Placement of synapse types

The `synTypes` nested tuple defines the placement of patches for different synapse models. Default layout is

AMPA	NMDA
GABA_A	GABA_B

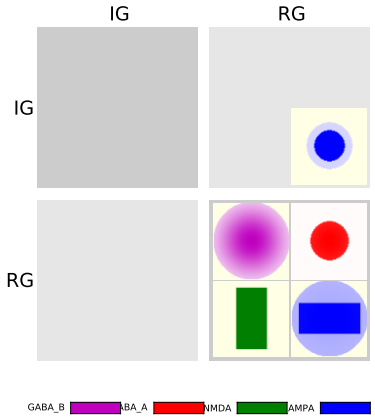
All four matrix elements are shown in this layout only when using `mode='layer'` display. Otherwise, one or the other row is shown. Note that synapses that can arise from a layer simultaneously, must always be placed on one matrix row, i.e., in one group. As an example, we now invert placement, without any other changes:

```
467
468 cinv_syns = ( (cpl.SynType('GABA_B', -1, 'm'), cpl.SynType('GABA_A', -1, 'r'
    )),
469                 (cpl.SynType('NMDA'  ,  1, 'g'), cpl.SynType('AMPA'  ,  1, 'b'
                    )) )
470 cinv_cp = cpl.ConnectionPattern(c_layer, c_conn, synTypes=cinv_syns)
471 cinv_cp.plot()
472 pylab.show()
```



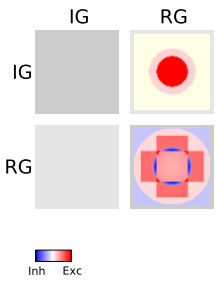
Notice that on each row the synapses are exchanged compared to the original figure above. When displaying by layer, also the rows have traded place:

```
474 cinv_cp.plot(aggrGroups=True)
475 pylab.show()
```



Totals are not affected:

```
478 cinv_cp.plot(aggrGroups=True,aggrSyns=True)
479 pylab.show()
```



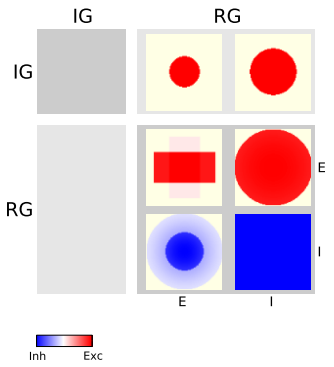
Weighting of synapse types in `totals` mode

Different synapses may have quite different efficacies, so weighting them all with +-1 when computing totals may give a wrong impression. Different weights can be supplied as second argument to SynTypes(). We return to the normal placement of synapses and create two examples with very different weights:

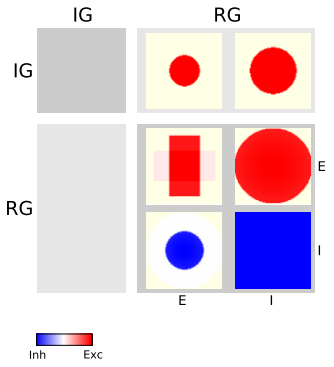
```
489 cw1_syns = ( (cpl.SynType('AMPA'   , 10, 'b'), cpl.SynType('NMDA'   ,   1, 'g'
        )),
490              (cpl.SynType('GABA_A', -2, 'g'), cpl.SynType('GABA_B', -10, 'b'
                 )) )
491 cw1_cp = cpl.ConnectionPattern(c_layer, c_conn, synTypes=cw1_syns)
492 cw2_syns = ( (cpl.SynType('AMPA'   ,   1, 'b'), cpl.SynType('NMDA'   ,  10, 'g
        ')),
493              (cpl.SynType('GABA_A', -20, 'g'), cpl.SynType('GABA_B',  -1, 'b
                 ')) )
494 cw2_cp = cpl.ConnectionPattern(c_layer, c_conn, synTypes=cw2_syns)
```

We first plot them both in population mode

```
496 cw1_cp.plot(aggrSyns=True)
497 pylab.show()
```

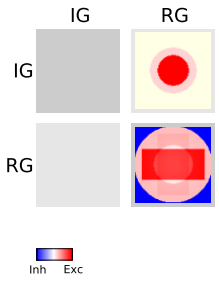



```
499
500 cw2_cp.plot(aggrSyns=True)
501 pylab.show()
```

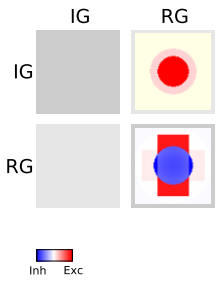


Finally, we plot them aggregating across groups and synapse models

```
503 cw1_cp.plot(aggrGroups=True,aggrSyns=True)
504 pylab.show()
```



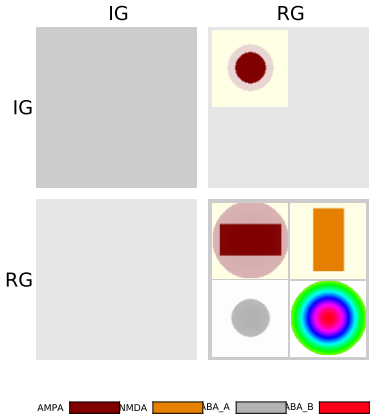
```
506
507 cw2_cp.plot(aggrGroups=True,aggrSyns=True)
508 pylab.show()
```



Alternative colors for synapse patches

Different colors can be specified using any legal color specification. Colors should be saturated, as they will be mixed with white. You may also provide a colormap explicitly. For this example, we use once more normal placement and weights. As all synapse types are shown in layer mode, we use that mode for display here.

```
517 cc_syns = ( (cpl.SynType('AMPA' , 1, 'maroon'), cpl.SynType('NMDA' , 1,
    (0.9,0.5,0) )),
518             (cpl.SynType('GABA_A', -1, '0.7' ), cpl.SynType('GABA_B', 1,
                    pylab.cm.hsv)) )
519 cc_cp = cpl.ConnectionPattern(c_layer, c_conn, synTypes=cc_syns)
520 cc_cp.plot(aggrGroups=True)
521 pylab.show()
```

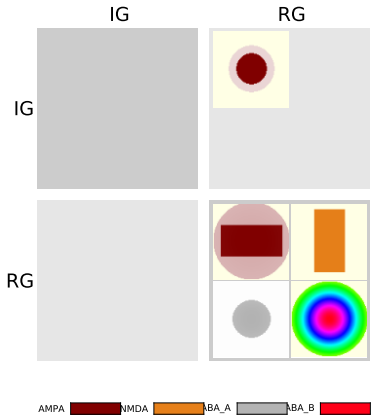


We get the following colors:

AMPA brownish NMDA golden orange GABA_A jet colormap from red (max) to blue (0) GABA_B grey

NB: When passing an explicit colormap, parts outside the mask will be shown to the "bad" color of the colormap, usually the "bottom" color in the map. To let points outside the mask appear in white, set the bad color of the colormap; unfortunately, this modifies the colormap.

```
532 pylab.cm.hsv.set_bad(cpl.colormaps.bad_color)
533 ccb_syns = ( (cpl.SynType('AMPA' , 1, 'maroon'), cpl.SynType('NMDA' , 1,
    (0.9, 0.5, 0.1))),
534             (cpl.SynType('GABA_A', -1, '0.7' ), cpl.SynType('GABA_B', 1,
                    pylab.cm.hsv)) )
535 ccb_cp = cpl.ConnectionPattern(c_layer, c_conn, synTypes=ccb_syns)
536 ccb_cp.plot(aggrGroups=True)
537 pylab.show()
```



Other configuration options

Some more adjustments are possible by setting certain module properties. Some of these need to be set before ConnectionPattern() is constructed.

Background color for masked parts of each patch

```
547 cpl.colormaps.bad_color = 'cyan'
```

Background for layers

```
550 cpl.plotParams.layer_bg = (0.8, 0.8, 0.0)
```

Resolution for patch computation

```
553 cpl.plotParams.n_kern = 5
```

Physical size of patches: longest egde of largest patch, in mm

```
556 cpl.plotParams.patch_size = 40
```

Margins around the figure (excluding labels)

```
559 cpl.plotParams.margins.left   = 40
560 cpl.plotParams.margins.top    = 30
561 cpl.plotParams.margins.bottom = 15
562 cpl.plotParams.margins.right  = 30
```

Fonts for layer and population labels

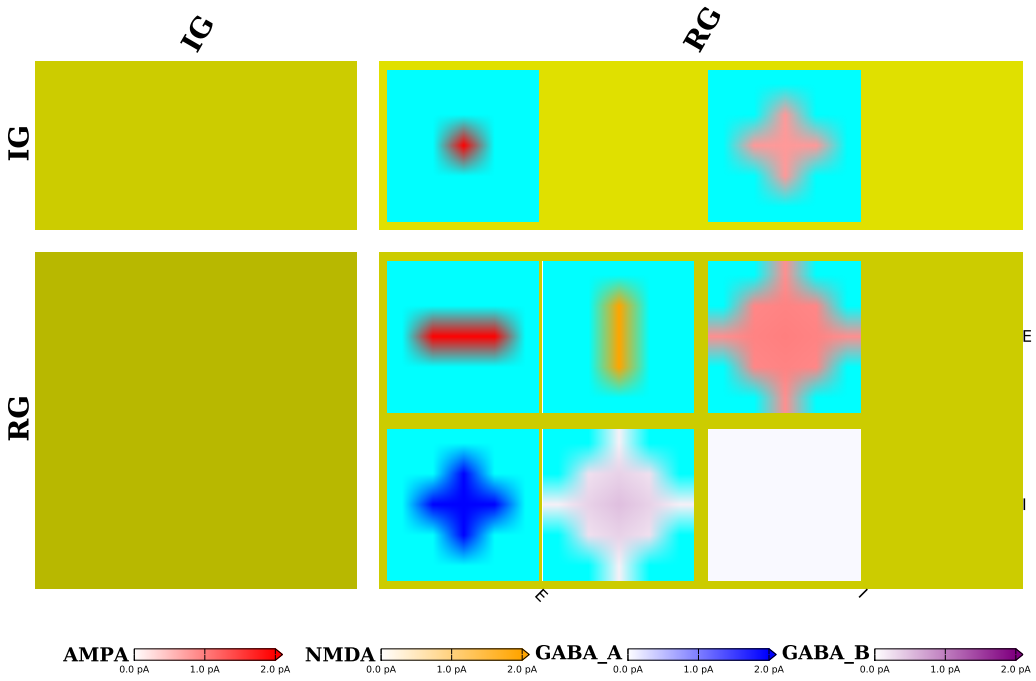
```
565 import matplotlib.font_manager as fmgr
566 cpl.plotParams.layer_font = fmgr.FontProperties(family='serif', weight='bold
    ', size='xx-large')
567 cpl.plotParams.pop_font   = fmgr.FontProperties('small')
```

Orientation for layer and population label

```
570 cpl.plotParams.layer_orientation = {'sender': 'vertical', 'target': 60}
571 cpl.plotParams.pop_orientation   = {'sender': 'horizontal', 'target': -45}
```

Font for legend titles and ticks, tick placement, and tick format

```
574 cpl.plotParams.legend_title_font = fmgr.FontProperties(family='serif',
    weight='bold', size='large')
575 cpl.plotParams.legend_tick_font = fmgr.FontProperties(family='sans-serif',
    weight='light', size='xx-small')
576 cpl.plotParams.legend_ticks = [0,1,2]
577 cpl.plotParams.legend_tick_format = '%.1f pA'
578
579 cx_cp = cpl.ConnectionPattern(c_layer, c_conn)
580 cx_cp.plot(colorLimits=[0,2])
581 pylab.show()
```



Several more options are available to control the format of the color bars (they all are members of plotPara

- legend_location : if 'top', place synapse name atop color bar
- cbwidth : width of single color bar relative to figure
- margins.colbar : height of lower margin set aside for color bar, in mm
- cbheight : height of single color bar relative to margins.colbar
- cbwidth : width of single color bar relative to figure width
- cbspace : spacing between color bars, relative to figure width
- cboffset : offset of first color bar from left margin, relative to figure width

You can also specify the width of the final figure, but this may not work well with on-screen display or here in pyreport. Width is in mm. Note that left and right margin combined are 70mm wide, so only 50mm are left for the actual CPT.

```
596 cx_cp.plot(fixedWidth=120)
597 pylab.show()
```