What’s new with

PyNN?

Sumatra?
What the **** is

PyNN?
Sumatra?
Simulator diversity: problem and opportunity

Cons

• Considerable difficulty in translating models from one simulator to another...
• ...or even in understanding someone else’s code.
• This:
  - impedes communication between investigators,
  - makes it harder to reproduce other people’s work,
  - makes it harder to build on other people’s work.

Pros

• Each simulator has a different balance between efficiency, flexibility, scalability and user-friendliness → can choose the most appropriate for a given problem.
• Any given simulator is likely to have bugs and hidden assumptions, which will be revealed by cross-checking results between different simulators → greater confidence in correctness of results.
# Find Models by Simulation Environment

Click on a link to show a list of models implemented in that simulation environment or programming language.

<table>
<thead>
<tr>
<th>Simulation Environment</th>
<th>Homepage</th>
<th>Number of models</th>
</tr>
</thead>
<tbody>
<tr>
<td>BioPAX (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Brian</td>
<td><img src="https://badgen.net/badge/4" alt="badge" /></td>
<td>4</td>
</tr>
<tr>
<td>C or C++ program</td>
<td><img src="https://badgen.net/badge/34" alt="badge" /></td>
<td>34</td>
</tr>
<tr>
<td>C or C++ program (web link to model)</td>
<td><img src="https://badgen.net/badge/19" alt="badge" /></td>
<td>19</td>
</tr>
<tr>
<td>CONTENT</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>CSIM</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>CSIM (web link to model)</td>
<td><img src="https://badgen.net/badge/3" alt="badge" /></td>
<td>3</td>
</tr>
<tr>
<td>CaIC Calcium Calculator</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>CaIC Calcium Calculator (web link to model)</td>
<td><img src="https://badgen.net/badge/7" alt="badge" /></td>
<td>7</td>
</tr>
<tr>
<td>Catacomb (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>CellExcite (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>CellML</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>CellML (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Chemesis</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>Dynamics Solver</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Emergent/PDP++</td>
<td><img src="https://badgen.net/badge/3" alt="badge" /></td>
<td>3</td>
</tr>
<tr>
<td>FORTRAN</td>
<td><img src="https://badgen.net/badge/4" alt="badge" /></td>
<td>4</td>
</tr>
<tr>
<td>FORTRAN (web link to a model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>GNUstep NeXTStep/OpenStep</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Genesis</td>
<td><img src="https://badgen.net/badge/13" alt="badge" /></td>
<td>13</td>
</tr>
<tr>
<td>Genesis (web link to model)</td>
<td><img src="https://badgen.net/badge/7" alt="badge" /></td>
<td>7</td>
</tr>
<tr>
<td>IChMASCOT</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>IGOR Pro</td>
<td><img src="https://badgen.net/badge/3" alt="badge" /></td>
<td>3</td>
</tr>
<tr>
<td>IonChannelLab</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Java</td>
<td><img src="https://badgen.net/badge/5" alt="badge" /></td>
<td>5</td>
</tr>
<tr>
<td>Java (web link to model)</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>KinNeSS (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>L-Neuron</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>MATLAB</td>
<td><img src="https://badgen.net/badge/67" alt="badge" /></td>
<td>67</td>
</tr>
<tr>
<td>MATLAB (web link to model)</td>
<td><img src="https://badgen.net/badge/34" alt="badge" /></td>
<td>34</td>
</tr>
<tr>
<td>MCell</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>MOOSE/PyMOOSE (web link to method)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>MVASpike</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>MedSim</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>NCS</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>NEST (formerly BLISS/SYNOD)</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>NEURON (web link to tool)</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>NSL</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Neosim</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Network</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>NeuGen</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Neuron</td>
<td><img src="https://badgen.net/badge/281" alt="badge" /></td>
<td>281</td>
</tr>
<tr>
<td>Neuron (web link to model)</td>
<td><img src="https://badgen.net/badge/14" alt="badge" /></td>
<td>14</td>
</tr>
<tr>
<td>NeuronC</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Neurongel (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Octave</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>PCSIM</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>PSPice</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>Pascal (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Pascal/Delphi</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>PyNN</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>Python</td>
<td><img src="https://badgen.net/badge/5" alt="badge" /></td>
<td>5</td>
</tr>
<tr>
<td>Python (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>QBasic/QuickBasic/Turbo Basic</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>QuB</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>R (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>SABER</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>SBML (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>SNNAP</td>
<td><img src="https://badgen.net/badge/21" alt="badge" /></td>
<td>21</td>
</tr>
<tr>
<td>SciLab</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>SciLab (web link to model)</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
<tr>
<td>Simulink</td>
<td><img src="https://badgen.net/badge/6" alt="badge" /></td>
<td>6</td>
</tr>
<tr>
<td>Spsice Symbolic SPICE</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Surf-Hippo</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Synthesis</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Topographya</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>Topographya (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>Virtual Caffe (web link to model)</td>
<td><img src="https://badgen.net/badge/3" alt="badge" /></td>
<td>3</td>
</tr>
<tr>
<td>XML (web link to model)</td>
<td><img src="https://badgen.net/badge/3" alt="badge" /></td>
<td>3</td>
</tr>
<tr>
<td>XNDC</td>
<td><img src="https://badgen.net/badge/0" alt="badge" /></td>
<td>0</td>
</tr>
<tr>
<td>XPP</td>
<td><img src="https://badgen.net/badge/50" alt="badge" /></td>
<td>50</td>
</tr>
<tr>
<td>XPP (web link to model)</td>
<td><img src="https://badgen.net/badge/7" alt="badge" /></td>
<td>7</td>
</tr>
<tr>
<td>neuroConstruct (web link to model)</td>
<td><img src="https://badgen.net/badge/1" alt="badge" /></td>
<td>1</td>
</tr>
<tr>
<td>parpale</td>
<td><img src="https://badgen.net/badge/2" alt="badge" /></td>
<td>2</td>
</tr>
</tbody>
</table>
Goal: write the code for a simulation once, run it on any supported simulator or hardware device without modification.

- keep the advantages of having multiple simulators or hardware devices
- but remove the translation barrier*.
Goal: write the code for a simulation once, run it on any supported simulator or hardware device without modification.

- keep the advantages of having multiple simulators or hardware devices
- but remove the translation barrier*. 

*a aka: having your cake and eating it

Cawan Cake by Nono Fara http://www.flickr.com/photos/n-o-n-o/3280580620/
sim.setup(timestep=0.1)
cell_parameters = {
    "tau_m": 12.0, "cm": 0.8, "v_thresh": -50.0,
    "v_reset": -65.0
}
pE = sim.Population((100, 100), sim.IF_cond_exp, cell_parameters,
    label="excitatory neurons")
pI = sim.Population((50, 50), sim.IF_cond_exp, cell_parameters,
    label="inhibitory neurons")
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution("normal", (10.0, 2.0))
input.rset("rate", rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
pE.inject(background)
pI.inject(background)
DDPC = sim.DistanceDependentProbabilityConnector
weight_distr = random.RandomDistribution("uniform", (0.0, 0.1))
connector = DDPC("exp(-d**2/400.0)", weights=weight_distr,
    delays="0.5+0.01d")
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5, tau_rec=800.0))
e2e = sim.Projection(pE, pE, connector, target="excitatory",
    synapse_dynamics=plasticity)
e2i = sim.Projection(pE, pI, connector, target="excitatory")
i2e = sim.Projection(pI, pE, connector, target="inhibitory")
import pyNN.neuron as sim

sim.setup(timestep=0.1)

cell_parameters = {
    "tau_m": 12.0, 
    "cm": 0.8, 
    "v_thresh": -50.0, 
    "v_reset": -65.0
}

pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
    label="excitatory neurons")

pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
    label="inhibitory neurons")

input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution("normal", (10.0, 2.0))
input.rset("rate", rate_distr)

background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
pE.inject(background)
pI.inject(background)

DDPC = sim.DistanceDependentProbabilityConnector

weight_distr = random.RandomDistribution("uniform", (0.0, 0.1))
connector = DDPC("exp(-d**2/400.0)", weights=weight_distr,
    delays="0.5+0.01d")

TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
e2e = sim.Projection(pE, pE, connector, target="excitatory",
    synapse_dynamics=plasticity)
e2i = sim.Projection(pE, pI, connector, target="excitatory")
i2e = sim.Projection(pI, pE, connector, target="inhibitory")
import pyNN.nest as sim

sim.setup(timestep=0.1)

cell_parameters = {
    "tau_m": 12.0, 
    "cm": 0.8, 
    "v_thresh": -50.0, 
    "v_reset": -65.0
}

pE = sim.Population((100, 100), sim.IF_cond_exp, cell_parameters,
                     label="excitatory neurons")

pI = sim.Population((50, 50), sim.IF_cond_exp, cell_parameters,
                     label="inhibitory neurons")

input = sim.Population(100, sim.SpikeSourcePoisson)

rate_distr = random.RandomDistribution("normal", (10.0, 2.0))

input.rset("rate", rate_distr)

background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)

pE.inject(background)

pI.inject(background)

DDPC = sim.DistanceDependentProbabilityConnector

weight_distr = random.RandomDistribution("uniform", (0.0, 0.1))

connector = DDPC("exp(-d**2/400.0)", weights=weight_distr,
                  delays="0.5+0.01d")

TMM = sim.TsodyksMarkramMechanism

depressing = sim.DynamicSynapse(fast=TMM(U=0.5, tau_rec=800.0))

e2e = sim.Projection(pE, pE, connector, target="excitatory",
                     synapse_dynamics=plasticity)

e2i = sim.Projection(pE, pI, connector, target="excitatory")

i2e = sim.Projection(pI, pE, connector, target="inhibitory")
import pyNN.hardware.facets.stage1 as sim
sim.setup(timestep=0.1)
cell_parameters = {"tau_m": 12.0, "cm": 0.8, "v_thresh": -50.0, "v_reset": -65.0}
pE = sim.Population((100, 100), sim.IF_cond_exp, cell_parameters,
label="excitatory neurons")
pI = sim.Population((50, 50), sim.IF_cond_exp, cell_parameters,
label="inhibitory neurons")
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution("normal", (10.0, 2.0))
input.rset("rate", rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
pE.inject(background)
pI.inject(background)
DDPC = sim.DistanceDependentProbabilityConnector
weight_distr = random.RandomDistribution("uniform", (0.0, 0.1))
connector = DDPC("exp(-d**2/400.0)", weights=weight_distr,
delays="0.5+0.01d")
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
e2e = sim.Projection(pE, pE, connector, target="excitatory",
synapse_dynamics=plasticity)
e2i = sim.Projection(pE, pI, connector, target="excitatory")
i2e = sim.Projection(pI, pE, connector, target="inhibitory")
1. **Spikes, membrane potential and synaptic conductances can now be saved to file in various binary formats.** To do this, pass a PyNN File object to Population.print_X(), instead of a filename. There are various types of PyNN File object, defined in the recording.files module, e.g., StandardTextFile, PickleFile, NumpyBinaryFile, HDF5ArrayFile.

2. **Added a reset() function and made the behaviour of setup() consistent across simulators.** reset() sets the simulation time to zero and sets membrane potentials to their initial values, but does not change the network structure. setup() destroys any previously defined network.

3. **The possibility of expressing distance-dependent weights and delays was extended to the AllToAllConnector and FixedProbabilityConnector classes.** To reduce the number of arguments to the constructors, the arguments affecting the spatial topology (periodic boundary conditions, etc.) were moved to a new Space class, so that only a single Space instance need be passed to the Connector constructor.

4. **Assorted speed-ups**

5. **Testing** that results are independent of number of processors added to regression tests.
Since CodeJam #3

1. internal sub-package reorganisation in preparation for multi-compartmental models
2. connection speed-ups
3. added SmallWorldConnector
4. removed v_init as a parameter, replaced with Population.initialize() method
5. Population structure no longer restricted to a grid
6. began implementing PopulationView (sub-populations) and Assembly (collection of Populations)
### Active Tickets by Milestone

This report shows how to color results by priority, while grouping results by milestone.

Last modification time and description are included as hidden fields for useful RSS export.

#### 0.7.0 Release

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Summary</th>
<th>Component</th>
<th>Version</th>
<th>Type</th>
<th>Owner</th>
<th>Created</th>
<th>Reporter</th>
</tr>
</thead>
<tbody>
<tr>
<td>#155</td>
<td>When using <code>Projection.set/getWeights()</code> with arrays, the case of multiple connections between source and target is not consistently handled or tested</td>
<td>all</td>
<td>trunk</td>
<td>defect</td>
<td>adavidson</td>
<td>02/03/10</td>
<td>adavidson</td>
</tr>
<tr>
<td>#160</td>
<td>SpikeSourceArray ignores non-spike times</td>
<td>common</td>
<td>trunk</td>
<td>defect</td>
<td>mshnhcker</td>
<td>02/26/10</td>
<td>mshnhcker</td>
</tr>
<tr>
<td>#62</td>
<td>Draw dynamic synapses parameters from RNG Distributions</td>
<td>common</td>
<td>trunk</td>
<td>enhancement</td>
<td>pierre</td>
<td>04/03/08</td>
<td>pierre</td>
</tr>
<tr>
<td>#102</td>
<td>SubPopulation should be extracted from a Population object</td>
<td>all</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>05/21/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#113</td>
<td>Add <code>Projection.record_weights()</code> or something similar</td>
<td>all</td>
<td>trunk</td>
<td>enhancement</td>
<td>pierre</td>
<td>08/27/00</td>
<td>pierre</td>
</tr>
<tr>
<td>#119</td>
<td>Pacing and cells labeling</td>
<td>all</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>11/04/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#137</td>
<td>Reimplement connectors in terms of arrays</td>
<td>common</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>05/05/00</td>
<td>adavidson</td>
</tr>
<tr>
<td>#158</td>
<td>All issues of which connections are local should be dealt with within ConnectionManager, not in the Connectors</td>
<td>common</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>05/26/00</td>
<td>adavidson</td>
</tr>
<tr>
<td>#159</td>
<td>A way to reset the list of things to record</td>
<td>common</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>01/26/10</td>
<td>adavidson</td>
</tr>
<tr>
<td>#154</td>
<td>Implement <code>Population.record_asyn()</code> method and <code>record_asyn()</code> function in the <code>pscim</code> module</td>
<td>pscim</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>02/02/10</td>
<td>adavidson</td>
</tr>
<tr>
<td>#161</td>
<td>Rethink the connectors as convergent_connect() and problems with set/getWeight</td>
<td>Documentation</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>01/25/10</td>
<td>adavidson</td>
</tr>
<tr>
<td>#23</td>
<td>Separate standard models into a membrane part and a synapse part</td>
<td>unspec</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/07/00</td>
<td>adavidson</td>
</tr>
<tr>
<td>#4</td>
<td>Any int or float cell or connection parameter should be replaceable by a <code>RandomDistribution</code> object</td>
<td>unspec</td>
<td>trunk</td>
<td>task</td>
<td>pierre</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#5</td>
<td>More sophisticated parameter handling</td>
<td>unspec</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#16</td>
<td>Test multicompart models with 'neuron' module</td>
<td>unspec</td>
<td>trunk</td>
<td>task</td>
<td>adavidson</td>
<td>01/11/06</td>
<td>adavidson</td>
</tr>
<tr>
<td>#125</td>
<td>Update NMODL files in src/hoc to be thread-safe, where possible</td>
<td>hoc</td>
<td>trunk</td>
<td>task</td>
<td>adavidson</td>
<td>01/27/00</td>
<td>adavidson</td>
</tr>
<tr>
<td>#51</td>
<td>Split 'common' into multiple files</td>
<td>common</td>
<td>trunk</td>
<td>defect</td>
<td>somebody</td>
<td>02/16/10</td>
<td>adavidson</td>
</tr>
<tr>
<td>#124</td>
<td>setup.py does not recompile .moc files</td>
<td>noc</td>
<td>trunk</td>
<td>defect</td>
<td>somebody</td>
<td>03/15/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#165</td>
<td>Temporarily files are not closed correctly by Recorder class</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#24</td>
<td>Implement user-specified output formats for spikes, etc</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#35</td>
<td>Access to neuron variables via properties of the ID class</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#85</td>
<td>Add a <code>get_simulation()</code> function</td>
<td>noc</td>
<td>trunk</td>
<td>defect</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#60</td>
<td>Change <code>set(cells, params, val)</code> to <code>set(cells, **Parameters)</code></td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#114</td>
<td><code>dummy</code> or <code>test</code> module to check argument types, etc in code before starting a time-consuming run</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#132</td>
<td>The selection of dynamic synapse models that is done inoffset... would be better done in the <code>synapsesDynamics</code> class</td>
<td>noc</td>
<td>trunk</td>
<td>enhancement</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#133</td>
<td>Add functions <code>run_for()</code> and <code>run_until()</code> with <code>run()</code> as alias for <code>run_for()</code></td>
<td>noc</td>
<td>trunk</td>
<td>enhancement</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#139</td>
<td>Add a FromArrayConnector class</td>
<td>noc</td>
<td>trunk</td>
<td>enhancement</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#138</td>
<td>Thoroughly test with native models</td>
<td>noc</td>
<td>trunk</td>
<td>enhancement</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#42</td>
<td>When creating a Projection it is not clear how to define if it is an excitatory or inhibitory synapse (in the case of conductance based synapses)</td>
<td>noc</td>
<td>trunk</td>
<td>enhancement</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#164</td>
<td>In the 'random' module, we should use mpyo instead of relying on the correct rank, etc. being passed</td>
<td>noc</td>
<td>trunk</td>
<td>enhancement</td>
<td>somebody</td>
<td>03/16/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#1</td>
<td>Combine <code>print()</code> and <code>print_v()</code> into a single write() that takes what-to-print as an argument</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#70</td>
<td>Create a NEURON equivalent of EJ's adapting I&amp;F model in NEST</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#9</td>
<td>Add an Izhikevich standard model</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#10</td>
<td>Improve NEURON implementation of IF_curr_alpha, etc</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#20</td>
<td>Performance benchmarks</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#54</td>
<td>Thoroughly test with native models</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/24/07</td>
<td>adavidson</td>
</tr>
<tr>
<td>#67</td>
<td>Move the <code>max_decay</code> argument of <code>setup()</code> into <code>extra_params</code>, since not all simulators use k</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/05/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#73</td>
<td>Compatible output should save times in voltage files</td>
<td>noc</td>
<td>trunk</td>
<td>task</td>
<td>somebody</td>
<td>04/06/08</td>
<td>adavidson</td>
</tr>
<tr>
<td>#151</td>
<td>Additional information regarding python installation with NEURON</td>
<td>Documentation</td>
<td>trunk</td>
<td>enhancement</td>
<td>adavidson</td>
<td>12/08/09</td>
<td>bikienplan</td>
</tr>
</tbody>
</table>

Notes: See TracReports for help on using and creating reports.
“I thought I used the same parameters but I’m getting different results”
“I thought I used the same parameters but I’m getting different results”

“I can’t remember which version of the code I used to generate figure 6”
“I thought I used the same parameters but I’m getting different results”

“I can’t remember which version of the code I used to generate figure 6”

“The new student wants to reuse that model I published three years ago but he can’t reproduce the figures”
“I thought I used the same parameters but I’m getting different results”

“I can’t remember which version of the code I used to generate figure 6”

“The new student wants to reuse that model I published three years ago but he can’t reproduce the figures”

“It worked yesterday”
“I thought I used the same parameters but I’m getting different results”

“I can’t remember which version of the code I used to generate figure 6”

“The new student wants to reuse that model I published three years ago but he can’t reproduce the figures”

“It worked yesterday”

“Why did I do that?”
Why isn’t it easy to reproduce a computational experiment exactly?
What can we do about it?
An automated lab notebook to record every detail of our simulations/analyses.
An automated lab notebook to record every detail of our simulations/analyses

“systematic capture”
What do we need to record?
What do we need to record?

- the code that was run
- how it was run (parameter files, command-line options)
- the platform on which it was run
- why was it run?
- what was the outcome?
What should this automated lab notebook look like?
Different researchers, different workflows

- command-line
- GUI
- batch jobs
- solo or collaborative
- any combination of these for different components and phases of the project
Requirements

- automate as much as possible, prompt the user for the rest
- interact with version control systems (Subversion, Git, Mercurial, Bazaar, Perforce, ...)
- support serial, distributed, batch simulations/analyses
- link to data generated by the simulation/analysis
- support all and any (command-line driven) simulation/analysis programs
- support both local and networked storage of simulation/analysis records
Requirements

Be very easy to use, or only the very conscientious will use it.
Sumatra Simulation Management Tool

http://neuralensemble.org/trac/sumatra
Sumatra

Nothing to do with Java
- a Python package, sumatra, to enable automated recording of provenance information
- can be used directly in your own code
- or as the basis for interfaces
Current

- a command line interface, `smt`
- a web interface, `smtweb`
- integrated into NeuronVisio

Future

- could be integrated into other existing GUIs (neuroConstruct, Topographica, nrngui)
- or new desktop/web-based GUIs written from scratch
Dependencies

➢ Python bindings for your preferred version control system (pysvn, mercurial, PyGit)

➢ Django (only needed for web interface)
Installation

`easy_install sumatra`
Using sumatra within your own scripts

```python
import numpy
import sys

parameter_file = sys.argv[1]
parameters = {}
execfile(parameter_file, parameters)  # this way of reading parameters
# is not necessarily recommended

numpy.random.seed(parameters["seed"])  
distr = getattr(numpy.random, parameters["distr"])  
data = distr(size=parameters["n"])

output_file = "example.dat"
numpy.savetxt(output_file, data)
```
import numpy
import sys
import time
from sumatra.projects import load_simulation_project
from sumatra.parameters import build_parameters

project = load_simulation_project()
start_time = time.time()

parameter_file = sys.argv[1]
parameters = build_parameters(parameter_file)

sim_record = project.new_record(parameters=parameters,
                               main_file=__file__,
                               label="api_example",
                               reason="reason for running this simulation")

numpy.random.seed(parameters["seed"])
distr = getattr(numpy.random, parameters["distr"])
data = distr(size=parameters["n"])

output_file = "%s.dat" % sim_record.label
numpy.savetxt(output_file, data)

sim_record.duration = time.time() - start_time
sim_record.data_key = sim_record.datastore.find_new_files(sim_record.timestamp)
project.add_record(sim_record)

project.save()
$ cd myproject
$ smt init MyProject
$ python main.py default.param
$ python main.py default.param
$ smt run --simulator=python --main=main.py default.param
$ python main.py default.param
$ smt run --simulator=python --main=main.py default.param
$ smt configure --simulator=python --main=main.py
$ python main.py default.param
$ smt run --simulator=python --main=main.py default.param
$ smt configure --simulator=python --main=main.py
$ smt run default.param
$ smt list
default_20090930-174949
default_20090930-175111

$ smt list -l
--------------------------------------------------
Label      : default_20090930-174949
Reason     :
Outcome    :
Duration   : 0.0548920631409
Script     : MercurialRepository at /path/to/myproject
            rf9ab74313efe (main file is main.py)
Executable : Python (version: 2.6.2) at /usr/bin/python
Timestamp  : 2009-09-30 17:49:49.235772
Tags       :
            .
            .
            .
$ smt run --label=haggling --reason="determine whether the gourd is worth 3 or 4 shekels" romans.param
$ smt comment "apparently, it is worth NaN shekels."
Eureka! Nobel prize here we come.
$ smt tag "Figure 6"
$ smt run --reason="test effect of a smaller time constant" default.param tau_m=10.0
$ smt repeat haggling_2009101002
The simulation results match.
$ smt
Usage: smt <subcommand> [options] [args]

Simulation management tool, version 0.1

Available subcommands:
  init
  configure
  info
  run
  list
  delete
  comment
  tag
  repeat
  help
$ smtweb 8002 &
What’s new since CodeJam #3

• Released version 0.1 (not vaporware anymore)

• Michele Mattioni added git support

• RecordStore can now contain records from multiple projects/users

• Added HttpRecordStore* enabling storing records on a remote server

*aalmost finished
Sumatran orangutan

http://neuralensemble.org/trac/sumatra

@apdavison

http://www.andrewdavison.info