Transforming LEMS/NeuroML v2.0 models to & from other formats

Padraig Gleeson

Lab of Prof. R. Angus Silver
Department of Neuroscience, Physiology and Pharmacology
University College London
<include href = "NaConductance.nml"/>
<include href = "LeakConductance.nml"/>
<include href = "KConductance.nml"/>

<cell id = "TestCell_ChannelML">

  <notes>A Simple cell with HH channels specified in ChannelML</notes>

  <morphology id = "morphology_TestCell_ChannelML">

    <segment id = "0" name = "Soma">
      <proximal x = "0.0" y = "0.0" z = "0.0" diameter = "16.0"/>
      <distal x = "0.0" y = "0.0" z = "0.0" diameter = "16.0"/>
    </segment>

    <segmentGroup id = "Soma">
      <member segment = "0"/>
    </segmentGroup>

    <segmentGroup id = "all">
      <include segmentGroup = "Soma"/>
    </segmentGroup>

    <segmentGroup id = "soma_group">
      <include segmentGroup = "Soma"/>
    </segmentGroup>

  </morphology>

  <!-- Adding the biophysical parameters -->

  <biophysicalProperties id = "biophys">

    <membraneProperties>

      <channelDensity id = "NaConductance_all" ionChannel = "NaConductance" condDensity = "120.0 mS_per_cm2" erev = "50.0 mV"/>
      <channelDensity id = "LeakConductance_all" ionChannel = "LeakConductance" condDensity = "0.3 mS_per_cm2" erev = "-54.3 mV"/>
      <channelDensity id = "KConductance_all" ionChannel = "KConductance" condDensity = "36.0 mS_per_cm2" erev = "-77.0 mV"/>
      <spikeThresh value = "0 mV"/>
      <specificCapacitance value = "1.0 uF_per_cm2"/>
      <initMembPotential value = "-60.0 mV"/>

    </membraneProperties>

  </biophysicalProperties>

</cell>
lems/ChannelMLConvert/ChannelML2NeuroML2.xsl

enables mapping ChannelML -> NeuroML 2 for (simple) channels & synapses
NeuroML v2.0 → LEMS → LEMS interpreter

- NEURON
- Brian
- SBML
- CellML
- Python
- NineML
- Graphs of behaviour
NeuroML Development Workshop 2011

http://www.NeuroML.org

NeuroML v2.0

LEMS

LEMS interpreter

Graphs of behaviour

NEURON

Brian

SBML

CellML

Python

NineML
Adaptive Exponential Integrate & Fire neuron

- Brette & Gernstner 2005

\[
C \frac{dV}{dt} = -g_L(V - E_L) + g_L \Delta_T \exp \left( \frac{V - V_T}{\Delta_T} \right) - g_c(t)(V - E_c) - g_i(t)(V - E_i) - w
\]

\[
\tau_w \frac{dw}{dt} = a(V - E_L) - w
\]

At spike time ($V > 20$ mV): $V \rightarrow E_L$
$w \rightarrow w + b$
Example network in NeuroML 2.0

```xml
<adExIaFCell id="adExBurst2"  C="281pF"  gL="30nS"  E_L="-70.6mV"  reset="-48.5mV"  VT = "-50.4mV"
  thresh = "-40.4mV"  delT="2mV"  tauW="40ms"  a = "4nS"  b = "0.08nA"
  Iamp="0.8nA"  Idel="0ms"  Idur="2000ms"/>

<adExIaFCell id="adExRebound"  C="281pF"  gL="30nS"  E_L="-60mV"  reset="-51mV"  VT = "-54mV"
  thresh = "-30mV"  delT="2mV"  tauW="150ms"  a = "200ns"  b = "0.1nA"
  Iamp="-0.5nA"  Idel="150ms"  Idur="50ms"/>

<network id="net1">
  <population id="adExPop1"  component="adExBurst2"  size="1"/>
  <population id="adExPop4"  component="adExRebound"  size="1"/>
</network>
```
Network to be simulated

```
Net1

adExPop1
adExPop2
```

Components

```
adExLaFCell (id = adExBurst2)
C = 2.81E-10 F, gL = 3.0E-8 S, EL = -0.0706 V,
VT = -0.054 V, thresh = -0.0404 V, reset = -0.0485 V,
delT = 0.0020 V, tauw = 0.04 s, Iamp = 5.0E-10 A,
Idel = 0 s, Idur = 2 s, a = 4.0E-9 S,
b = 8.0E-11 A
```

```
adExLaFCell (id = adExRebound)
C = 2.81E-10 F, gL = 3.0E-8 S, EL = -0.06 V,
VT = -0.054 V, thresh = -0.03 V, reset = -0.051 V,
delT = 0.0020 V, tauw = 0.15 s, Iamp = 5.0E-10 A,
Idel = 0.15 s, Idur = 0.05 s, a = 2.0E-7 S,
b = 1.0E-10 A
```

Component Types

```
adExLaFCell
w (current)
I (current)
v (voltage)

C (capacitance), gL (conductance), EL (voltage),
VT (voltage), thresh (voltage), reset (voltage),
delT (voltage), tauw (time), Iamp (current),
Idel (time), Idur (time), a (conductance),
b (current)

Isyn = synapses[i] (compute)
```

```
v' = (-1*gL*(v-EL) + gL*delT*exp(v-VT)/delT - w + Isyn)/C
w' = (a*(v-EL) - w) / tauw

IF (v > thresh) THEN
    (v = reset) AND (w = w+b)
IF ((t > Idel) AND (t < (Idel + Idur))) THEN
    (I = Iamp)
IF (t > (Idel + Idur)) THEN
    (I = 0)
```

```
abstractCellMemPot
v (voltage)
```

```
abstractCell
```

Parameter Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>281 pF</td>
</tr>
<tr>
<td>gL</td>
<td>30 nS</td>
</tr>
<tr>
<td>E_L</td>
<td>-70.6 mV</td>
</tr>
<tr>
<td>V_T</td>
<td>-50.4 mV</td>
</tr>
<tr>
<td>Delta_T</td>
<td>2 mV</td>
</tr>
<tr>
<td>tauw</td>
<td>144 ms</td>
</tr>
<tr>
<td>a</td>
<td>4 nS</td>
</tr>
<tr>
<td>b</td>
<td>0.0805 nA</td>
</tr>
</tbody>
</table>

```
\[
\frac{dV}{dt} = -g_L(V - E_L) + g_L \Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right) - g_L(t)(V - E_L) - g_L(t)(V - E_L) - w
\]

```
\[
\frac{dw}{dt} = a(V - E_L) - w
\]

At spike time (V > 20 mV): V \rightarrow EL 

w \rightarrow w + b
Network to be simulated

net1

fnPop1

1

Components

f FitzHugNagumoCell (id = fn1)

I = 0.8

Component Types

f FitzHugNagumoCell

V (none)
W (none)
I (none)
SEC (time) == 1s

V' = (V - V^3 - W + I) / SEC
W' = (0.08 * (V + 0.7 - W)) / SEC

abstractCell
NineML

- NineML (Network Interchange format for NEuroscience) is being developed as part of the INCF Multiscale Modelling Program
- Introduced by Andrew Davison
- Language for describing large scale models of spiking neurons

- LEMS can export simple (e.g. Izhikevich cell) models as NineML
- The Python API for LEMS can be used to import NineML via libnineml
9ML example exported to 3 formats: LEMS, NEURON & SBML
Python API for LEMS

```python
def print "Building simulation using Python API for LEMS"
lem_model = lems.Lems()

comp = lems.Component(NML2StdCompTypes.adExIAFCell, "burster", C="281pF", "
gL="30nS", EL="-70.6mV", reset="-47.2mV", VT = "-50.4mV", 
thresh = "-20.4mV", delT="2mV", tauw="40ms", 
a ="4nS", b = "0.08nA", Iamp="0.8nA", Idel="0ms", Idur="2000ms"

lem_model.add_component(comp)
net = lems.Network("Network1")
net.add_population(lem.Population("pop1", comp.id, 1))
lem_model.add_network(net)
lem_model.gen_sim_with_default_plots(net.id, 250, 0.01)
lem_model.write_lems_and_run("AdEx.xml")```
Python API for LEMS
Van der Pol oscillator

Analysis

When $x$ is small, the quadratic term $x^2$ is negligible and the system becomes a linear differential equation with a negative damping $-\epsilon \dot{x}$. Thus, the fixed point $(x = 0, \dot{x} = 0)$ is unstable (an unstable focus when $0 < \epsilon < 2$ and an unstable node, otherwise). On the other hand, when $x$ is large, the term $x^2$ becomes dominant and the damping becomes positive. Therefore, the dynamics of the system is expected to be restricted in some area around the fixed point. Actually, the van der Pol system (1) satisfies the Liénard's theorem ensuring that there is a stable limit cycle in the phase space. The van der Pol system is therefore a Liénard system.

Using the Liénard's transformation $y = x - x^3/3 - \dot{x}/\epsilon$, equation (1) can be rewritten as

$$\dot{x} = \epsilon \left(x - \frac{1}{3}x^3 - y\right)$$

and

$$\dot{y} = \frac{\dot{x}}{\epsilon}$$

which can be regarded as a special case of the FitzHugh-Nagumo model (also known as Bonhoeffer-van der Pol model).

Small Damping

When $\epsilon \ll 1$, it is convenient to rewrite equation (1) as

$$\dot{x} = \epsilon \left(x - \frac{1}{3}x^3\right) - y$$

and

$$\dot{y} = x$$

where the transformation $y = \epsilon(x - x^3/3) - \dot{x}$ was used. When $\epsilon = 0$, the system preserves the energy and has the solution $x = A \cos(t + \phi)$ and $y = A \sin(t + \phi)$. To obtain the approximated solution for small $\epsilon$, new variables $(u, v)$ which rotate with the unperturbed solution, i.e.,

$$u = x \cos t + y \sin t$$

$$v = -x \sin t + y \cos t$$

Figure 2: Change in $x$ over time for $\epsilon = 0.1$ with $x(0) = 0.5$ and $y(0) = 0$. 

$$
\begin{align*}
\dot{x} &= \epsilon \left(x - \frac{1}{3}x^3 - y\right) \\
\dot{y} &= \frac{\dot{x}}{\epsilon}
\end{align*}$$
Van der Pol oscillator

```python
print "Building simulation using Python API for LEMS"

lems_model = lems.Lems(include_neuroml2_types=False)

# Define ComponentType based on http://www.scholarpedia.org/article/Van_der_Pol_oscillator
comp_type = lems.ComponentType("vanderPolOscillator")

comp_type.parameters.append(lems.Parameter("epsilon"))

comp_type.behaviors[0].add_state_variable(lems.StateVariable("x"))
comp_type.behaviors[0].add_state_variable(lems.StateVariable("y"))

init = lems.OnStart()
init.state_assignments.append(lems.StateAssignment("x", "0.5"))
comp_type.behaviors[0].on_starts.append(init)

comp_type.behaviors[0].time_derivatives.append(lems.TimeDerivative("x", "epsilon * (x - (x^3)/3 -y)"))
comp_type.behaviors[0].time_derivatives.append(lems.TimeDerivative("y", "x/epsilon"))

lems_model.add_component_type(comp_type)

# Add instance of ComponentType
comp = lems.Component(comp_type.name, "dampedOscillator", epsilon="0.1")
lems_model.add_component(comp)

lems_model.gen_sim_with_default_plots(comp.id, 100000, 1)  # 100 seconds
lems_model.write_lems_and_run("vanderPol.xml")
```
Van der Pol oscillator

Network to be simulated

dampedOscillator

Components

Component (id = dampedOscillator)
epsilon = 0.1

Component Types

vanderPolOscillator
x (none)
y (none)
epsilon (none)
x' = epsilon * (x - (x^3)/3 - y)
y' = x/epsilon
SBML support

- Export of simple models
  - Izhikevich
  - I & F
  - Adaptive exponential
- Import of many SBML models into LEMS
  - Can be reused as ComponentTypes for any LEMS simulation
SBML export
SBML import

<table>
<thead>
<tr>
<th>Curation result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-08-10T17:20:50+00:00</td>
</tr>
<tr>
<td>Comment: Figure 2 RS, IB, CH, FS, LTS have been simulated by MathSBML.</td>
</tr>
</tbody>
</table>


Graphs showing voltage (v) and current (i) over time for different parameters.
SBML import
SBML support

- Enabled by JSBML

- SBML Test Suite
  - 327 out of 947 SBML models successful matching target data...
NEURON

Brian

SBML

CellML

LEMS

LEMS interpreter

Graphs of behaviour

Python

NineML

NeuroML v2.0

NeuroML v2.0