Introducing NESTML

NESTML is a domain-specific language for neuron and synapse models.

Using PyNEST, you instantiate and connect the models that you define in NESTML.
Introducing NESTML

• Concise; low on boilerplate
• Speak in the vernacular of the neuroscientist (keywords such as neuron, synapse)
• Easy (dynamical) equation handling coupled with imperative-style programming
  \(\text{if } V_m \geq V_{\text{threshold}}: \ldots\)

NESTML comes with a code generation toolbox.

• Code generation (model definition but not instantiation)
• Automated ODE analysis and solver selection
• Flexible addition of targets using Jinja2 templates
Mapping biological neurons to NESTML

neuron-model based on an RC-circuit

neuron rc_neuron:

end

<<rc_neuron.nestml>>
neuron rc_neuron:

initial_values:
V_m mV = 0 mV
end

equations:
V_m' = -V_m/\tau_m + I_{syn}/C_m
end

parameters:
# values taken from experiments
C_m pF = 250 pF
tau_m ms = 10 ms
I_syn pA = 10 pA
end

update:
integrate_odes()
end
Simulating \textit{rc\_neuron}

- Simulating \textit{rc\_neuron} for 1000 ms with constant input current of 10 pA

→ Strictly positive membrane potential
→ No spikes
Adding the resting potential $E_L$

- Shift $V_m$ by $E_L$:

```
neuron rc_neuron_rest:  
  initial_values:  
    $V_m\ mV = E_L$
  end

  equations:  
    $V_m' = -(V_m-E_L)/\tau_m + I_{syn}/C_m$
  end

  parameters:  
    $E_L\ mV = -70\ mV$
  end

...  
end

→ Still no spikes
neuron rc_fire:
    parameters:
        V_th mV = -55 mV - E_L
    end
    update:
        integrate_odes()
        if V_m >= V_th:
            V_m = V_reset
            emit_spike()
        end
    end
end
Refractoriness

**NESTML**

```nestml
neuron rc_refractory:
  parameters:
    ref_counts integer = 0
    ref_timeout ms = 2 ms
  end
  internals:
    timeout_ticks integer = steps(ref_timeout)
  end
  update:
    if ref_counts == 0:
      integrate_odes()
      if V_m >= V_th:
        emit_spike()
        ref_counts = timeout_ticks
        V_m = V_reset
      end
    else:
      ref_counts -= 1
    end
  end
end
```

**Diagram**

- Integrate
- Refractory
- Fire
- `integrates_odes()`
- `[V_m > V_th]`
- `set_refractory(2 ms)`
- `[not is_refractory()]`
Simulating $rc_{\text{refractory}}$
Input handling

(Source: Wulfram Gerstner, Werner M. Kistler, Richard Naud, Liam Paninski - Neuronal Dynamics From Single Neurons to Networks and Models of Cognition)
neuron rc_input:
  initial_values:
    $V_m \text{ mV} = E_L$
  end

  equations:
    $V_m' = -(V_m-E_L)/\tau_m + I_{syn}/C_m$
  end

  parameters:
    $E_L \text{ mV} = -70 \text{ mV}$
... end

  input:
    $I_{syn} \text{ pA} <\text{ spike}$
  end

  output: spike
end

buffer can be inhibitory, excitatory or both (if nothing else stated)
neuron rc_alpha_response:
  initial_values:
    V_m mV = E_L
    I_a pA = 0 pA
    I_a' pA/ms = e/tau_syn * pA
  end

equations:
  shape I_a'' = -2 * I_a' / tau_syn - I_a / tau_syn**2
  V_m' = -(V_m-E_L)/tau_m + convolve(I_a, spikes)/C_m
  end

input:
  spikes pA <- spike
end

output: spike

update:
  integrate_odes()
  ...
end

end
Simulating $rc\_alpha\_response$
Shape notation

```nestml
neuron rc_alpha_response_shape:
    state:
        V_m mV = E_L
    end

    equations:
        shape I_a = (e/tau_syn) * t * exp(-t/tau_syn)
        V_m' = -(V_m-E_L)/tau_m + convolve(I_a, spikes)/C_m
    end

    input:
        spikes pA <- spike
    end

    output: spike

    update:
        integrate_odes()
        ...
    end

end
```

initial values computed automatically
Injecting currents

```python
currents = nest.Create('dc_generator', 1,
    {'amplitude': 100.0})
nest.Connect(currents, rc_neuron)
```

```python
neuron rc_neuron:

... 

equations:

    function I_syn pA = I_e + convolve(I_a, spikes) + currents
    V_m' = -V_m/tau_m + I_syn/C_m

end

input:

    currents pA <- current
    spikes pA <- spike

end

output: spike

... 

end
```

PyNEST

NEST

<<Runtime>>
import nest

nest.Install("nestmlmodels")

neuron = Create("rc_neuron")
nest.SetStatus(neuron, {"V_m":-72.0, "C_m":300.0})

mm = nest.Create('multimeter')
nest.SetStatus(mm, {"record_from":["V_m"]})
Connect(mm, neuron)

neuron rc_neuron:

  initial_values:
  V_m mV = -70mV
  end

  equations:
  V_m' = -(V_m-70mV)/tau_m + I_syn/C_m
  end

  parameters:
  C_m pF = 250pF
  tau_m ms = 10ms
  I_syn pA = 10pA
  end

end
Practical exercise: Izhikevich model

- Izhikevich: simple model for spiking neurons
  \[ v' = 0.04v^2 + 5v + 140 - u + I \]
  \[ u' = a(bv - u) \]

  if \( v \geq 30mV \) then
  \[
  \begin{cases} 
  v = c \\
  u = u + d 
  \end{cases}
  \]

- Tutorial task:
  1. Finish the model
  2. Change parameters to produce chattering behaviour

- See E. Izhikevich, IEEE Transactions on Neural Networks (2003) 14:1569-1572
Practical exercise: Izhikevich model

- Go to [https://jupyter.cscs.ch/](https://jupyter.cscs.ch/)
- Under "Piz Daint node type", select "mc" (for multicore) and click "Spawn"
- On the welcome screen, scroll down and click the "Terminal" button
- `git clone https://github.com/jougs/HPAC_Training --depth=1 && pip3 install nestml --user && cp HPAC_Training/.jupyterhub.env ~`
- Restart the Jupyter server:
  - Go to File → Hub control panel → "Stop my server"
  - Click "Start my server" → "Launch server" and use the same details as in the first step
- Now, everything is installed and we are ready to go! Double click the "HPAC_Training" folder in the left panel, then open the "NESTML" folder and open the notebook "NESTML-izhikevich-tutorial"
- Step through the notebook and convince yourself that everything works. Change `izhikevich_solution.nestml` to `izhikevich_task.nestml`, and finish the model!