Introduction to NESTML

Tom Tetzlaff

Institute of Neuroscience and Medicine (INM-6), Jülich Research Centre and JARA

t.tetzlaff@fz-juelich.de

http://www.csn.fz-juelich.de

EITN fall school, Paris, 22.09.2023

https://github.com/tomtetzlaff/2023_eitnfallschool

Outline

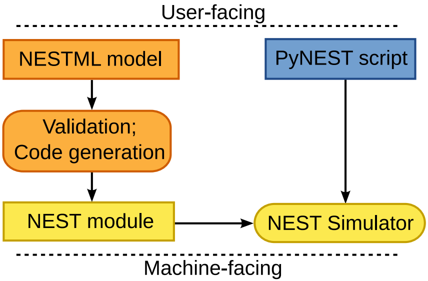
Overview

“Hello world!”

Example

Overview

- domain-specific language for definition of **custom neuron and synapse models**
- specifically tailored for NEST (SpiNNaker and NEST GPU support in progress)
- NESTML toolchain includes
  - syntax validation
  - system analysis and automatized selection of appropriate solving method (using ODE-toolbox)
  - code generation (C++ for NEST)
- see NESTML language concepts



- code: `https://github.com/nest/nestml`
- docs: `https://nestml.readthedocs.io`

nest::ml

“Hello world!”: NESTML neuron model definition I

```

neuron iaf_psc_exp:

  state:
    r integer = 0      # refratory state
    V_m mV = 0 mV      # membrane potential

  equations:
    kernel I_kernel_exc = exp(-t / tau_syn_exc)
    kernel I_kernel_inh = exp(-t / tau_syn_inh)
    recordable inline I_syn pA = convolve(I_kernel_exc, ExcInput) * pA - convolve(I_kernel_inh, In

    V_m' = - (V_m - E_L) / tau_m + (I_syn + I_e + IStim) / C_m

  parameters:
    C_m pF = 250 pF           # membrane capacitance
    tau_m ms = 10 ms          # membrane time constant
    tau_syn_exc ms = 5 ms     # time constant of excitatory synapses
    tau_syn_inh ms = 5 ms     # time constant of inhibitory synapses
    t_ref ms = 2 ms           # refractory period
    E_L  mV = 0.0 mV          # resting potential
    V_reset mV = 0.0 mV       # reset potential
    V_th   mV = 15.0 mV       # spike threshold
    I_e pA = 0 pA             # constant external input current

  internals:
    RefractoryCounts integer = steps(t_ref) # refractory time in steps

  input:

```

“Hello world!”: NESTML neuron model definition II

```
    ExcInput <- excitatory spike
    InhInput <- inhibitory spike
    IStim     pA <- continuous

  output:
    spike

  update:

    integrate_odes()

    if r == 0:         # neuron is not refractory
      if V_m >= V_th:  # threshold crossing
        emit_spike()
        r = RefractoryCounts
        V_m = V_reset

    else:              # neuron is refractory
      V_m = V_reset
      r -= 1
```

(see `iaf_psc_exp.nestml`)

"Hello world!": PyNEST code

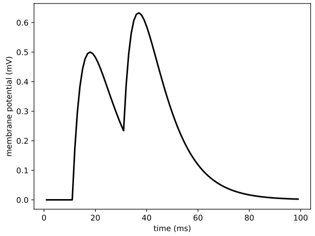
```python
import nest                                                          # import NEST module
import matplotlib.pyplot as plt                                      # for plotting
from pynestml.frontend.pynestml_frontend import generate_nest_target # NESTML

# compile nestml model (needs to be done only once)
generate_nest_target(input_path="../nestml/iaf_psc_exp.nestml",
                     target_path="./nestml_target",
                     suffix="_nestml",
                     logging_level='ERROR')

# install resulting NESTML module to make models available in NEST
nest.Install('nestmlmodule')

nest.ResetKernel() # reset simulation kernel

neuron=nest.Create('iaf_psc_exp_nestml') # create LIF neuron with exponential synaptic currents

...

plt.ylabel('membrane potential (mV)')
plt.savefig('./figures/hello_world_nestml.pdf')
```

(see `hello_world_nestml.py`)

"Hello world!": PyNEST code

```
import nest                                                         # import NEST module
import matplotlib.pyplot as plt                                     # for plotting
from pynestml.frontend.pynestml_frontend import generate_nest_target # NESTML

# compile nestml model (needs to be done only once)
generate_nest_target(input_path="../nestml/iaf_psc_exp.nestml",
                     target_path="./nestml_target",
                     suffix="_nestml",
                     logging_level='ERROR')

# install resulting NESTML module to make models available in NEST
nest.Install('nestmlmodule')

nest.ResetKernel() # reset simulation kernel

neuron=nest.Create('iaf_psc_exp_nestml') # create LIF neuron with exponential synaptic currents

...

plt.ylabel('membrane potential (mV)')
plt.savefig('./figures/hello_world_nestml.pdf')
```

hello_world_nestml.pdf:



(see hello_world_nestml.py)

Example: NESTML synapse model definition I

```
synapse stdp_pl:

  state:
    w           real = 1.    @nest::weight  # synaptic weight
    pre_trace   real = 0.                   # presynaptic trace
    post_trace  real = 0.                   # postsynaptic trace

  parameters:
    the_delay   ms   =  1    ms  @nest::delay
    lambda      real =  0.01
    alpha       real =  0.1
    tau_tr_pre  ms   = 15    ms
    tau_tr_post ms   = 30    ms
    mu_plus     real =  0.4
    w_0         real =  1.0

  equations:
    pre_trace'  = - pre_trace / tau_tr_pre
    post_trace' = - post_trace / tau_tr_post

  input:
    pre_spikes  <- spike
    post_spikes <- spike

  output:
    spike

  onReceive(post_spikes):
    post_trace += 1

```

Example: NESTML synapse model definition II

```
    # update for causal firing (potentiation)
    w = w + lambda * w_0 * (w/w_0)**mu_plus * pre_trace

  onReceive(pre_spikes):
    pre_trace += 1

    # update for acausal firing (depression)
    w_ real = w - alpha*lambda * w * post_trace

    # zero clipping
    w = max(0.0,w_)

    # deliver spike to postsynaptic partner
    deliver_spike(w, the_delay)

```

(see `stdp_pl_synapse.nestml`)

Example: PyNEST code I

```python
# compile nestml model (needs to be done only once)
generate_nest_target(input_path=["../nestml/iaf_psc_exp.nestml",
                                 "../nestml/stdp_pl_synapse.nestml"],
                     target_path="./nestml_target",
                     logging_level='ERROR',
                     suffix="_nestml",
                     codegen_opts = {"neuron_synapse_pairs": [{"neuron": "iaf_psc_exp",
                                                               "synapse": "stdp_pl",
                                                               "post_ports": ["post_spikes"]}]}
)

# install resulting NESTML module to make models available in NEST
nest.Install('nestmlmodule')

# after this, the following two models are available in NEST
neuron_model_name = 'iaf_psc_exp_nestml__with_stdp_pl_nestml'
synapse_model_name = 'stdp_pl_nestml__with_iaf_psc_exp_nestml'

...

pre_neuron=nest.Create('parrot_neuron') # create presynaptic neuron
post_neuron=nest.Create(neuron_model_name,2) # create postsynaptic neuron

...
```

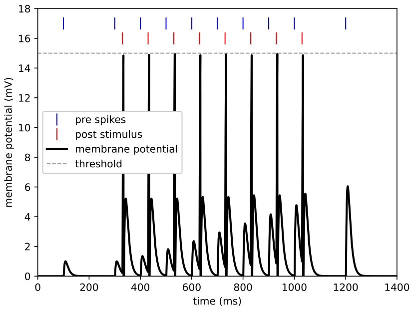
Example: PyNEST code II

```
# configure STDP synapse
nest.CopyModel(synapse_model_name,"plastic_synapse", {
    "weight":      100.0,   # initial synaptic weight (pA)
    "delay":         0.1,   # spike transmission delay (ms)
    'lambda':       10.0,   # (dimensionless) learning rate for causal updates
    'tau_tr_pre':  100.0,   # time constant of presynaptic trace (ms)
    'tau_tr_post': 100.0,   # time constant of postsynaptic trace (ms)
})

# connect pre neuron and post neuron with the STDP synapse
nest.Connect(pre_neuron, post_neuron, syn_spec="plastic_synapse")
```

...

(see `hello_world_plastic_nestml.py`)

`hello_world_nestml.pdf`:



Thanks