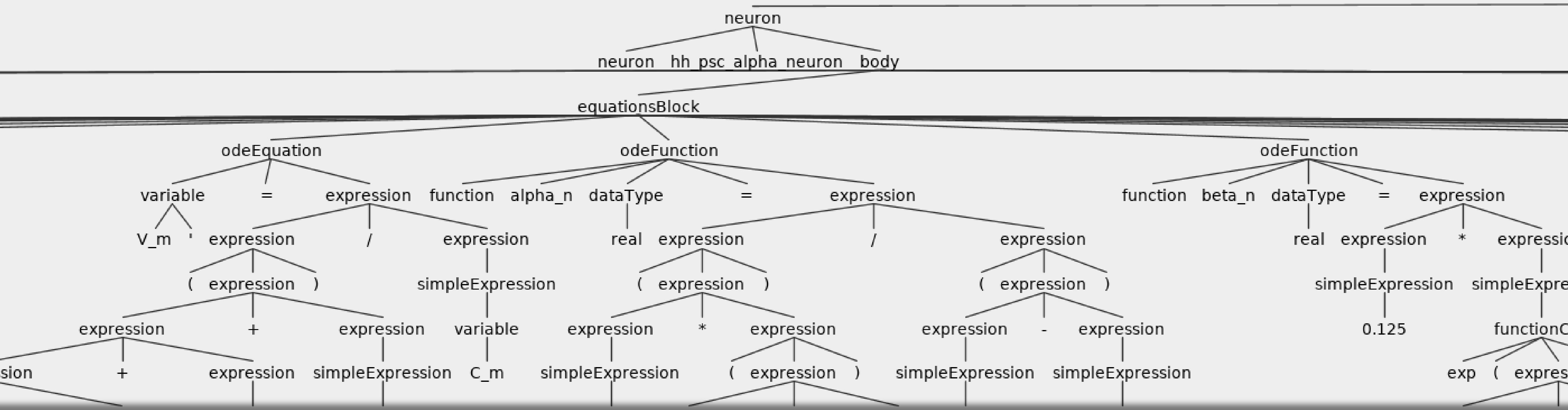
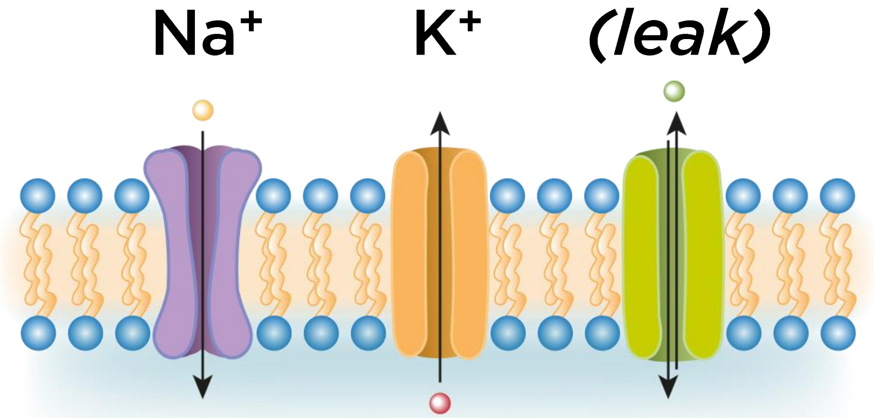
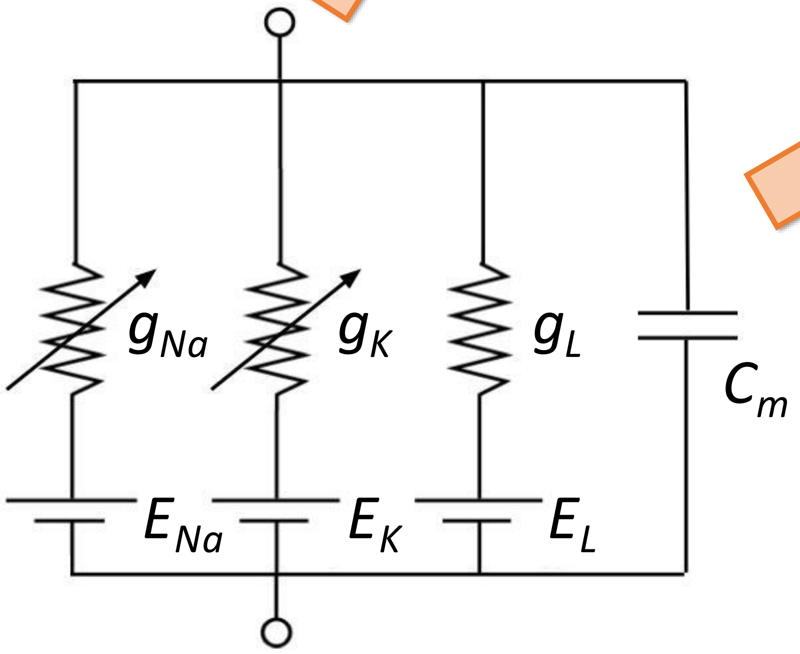


Neuron and synapse models in NESTML: From specification to simulation

Charl Linssen <c.linssen@fz-juelich.de> | HBP CodeJam 2021 | 2021 Nov 23rd

JÜLICH Forschungszentrum
Member of the Helmholtz Association






```
neuron hodgkin_huxley:
  state:
    V_m mV = -65 mV
    Act_m, Act_n, Inact_h ...
  end

  equations:
    shape syn_psc_kernel = exp(-t / tau_syn)
    inline I_Na pA = g_Na * Act_m**3 * Inact_h * (V_m - E_Na)
    inline I_K pA = ...
    inline I_L pA = g_L * (V_m - E_L)
    V_m' = -(I_Na + I_K + I_L) / C_m
           + convolve(syn_psc_kernel, spikes)
    Act_n' = (alpha_n * (1 - Act_n) - beta_n * Act_n) / ms
    Act_m' = ...
    Inact_h' = ...
  end

  parameters:
    C_m pF = 250 pF
    V_threshold mV = 40 mV
    ...
  end

  update:
    integrate_odes()
    if V_m >= V_threshold:
      emit_spike()
    end
  end
end
```

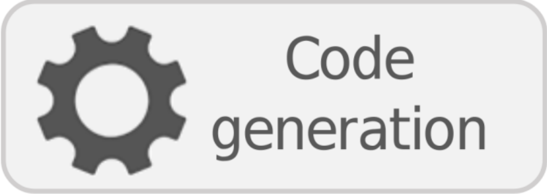
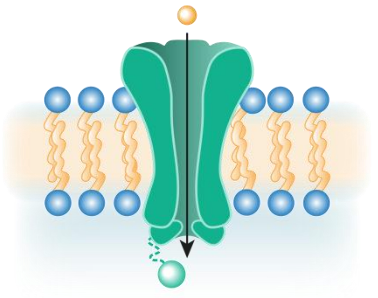
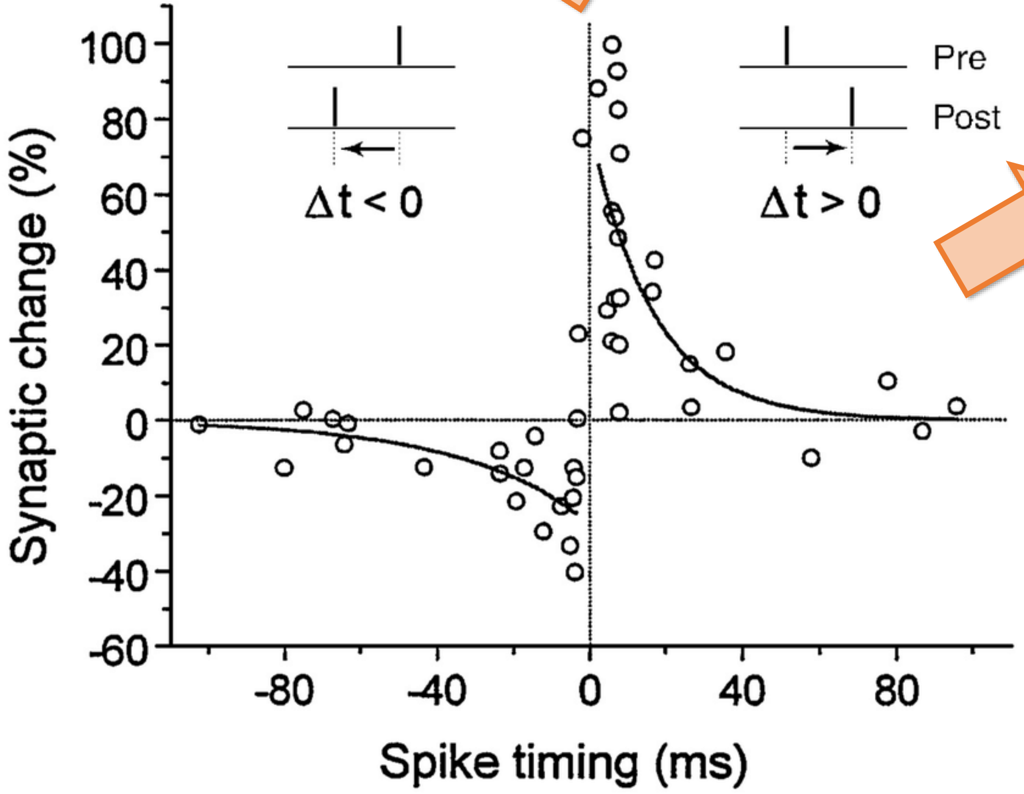


Image credit: (c) S.B.C. Lehmann, FZ Jülich, (m) C. Linssen

NMDA (Ca^{2+})





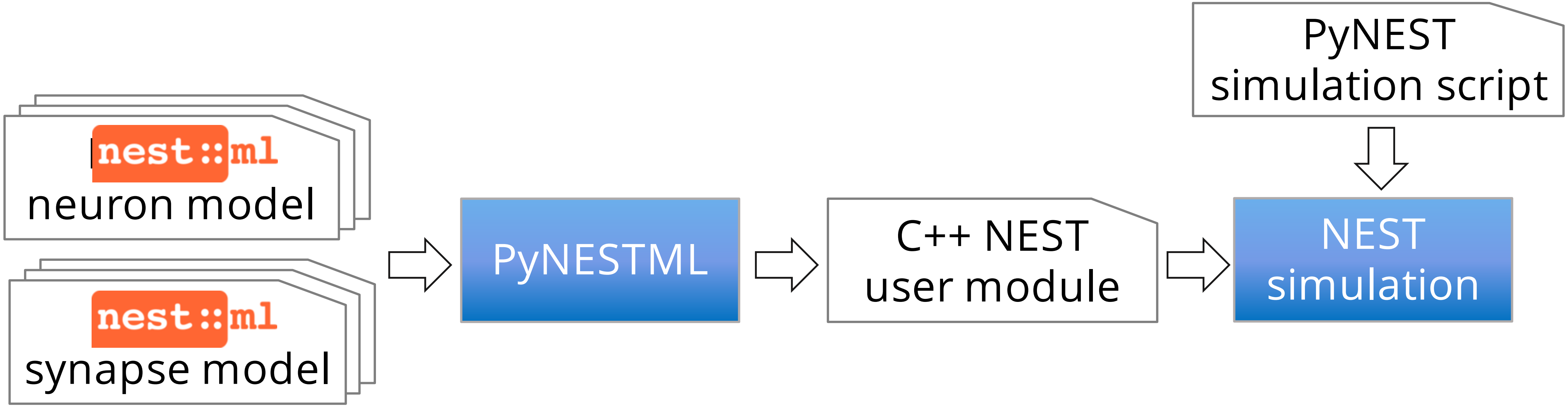

```
synapse stdp:
  state:
    w real = 1
    tr_post real = 0
    tr_pre real = 0
  end

  equations:
    tr_pre' = -tr_pre / tau_tr
    tr_post' = -tr_post / tau_tr
  end

  input:
    pre_spikes real <- spike
    post_spikes real <- spike
  end

  onReceive(pre_spikes):
    w -= alpha * tr_post        # depress synapse
    tr_pre += 1                 # update presynaptic trace
    deliver_spike(w, delay)     # to postsynaptic partner
  end

  onReceive(post_spikes):
    w += alpha * tr_pre         # potentiate synapse
    tr_post += 1                # update postsynaptic trace
  end

  parameters:
    delay ms = 1 ms             # dendritic delay
    tau_tr ms = 50 ms           # pre/post trace time const.
    alpha real = .02            # learning rate
  end
end
```

nest::ml

Image credit: (c) S.B.C. Lehmann, FZ Jülich, (m) C. Linssen; Bi & Poo 2001

Typical workflow

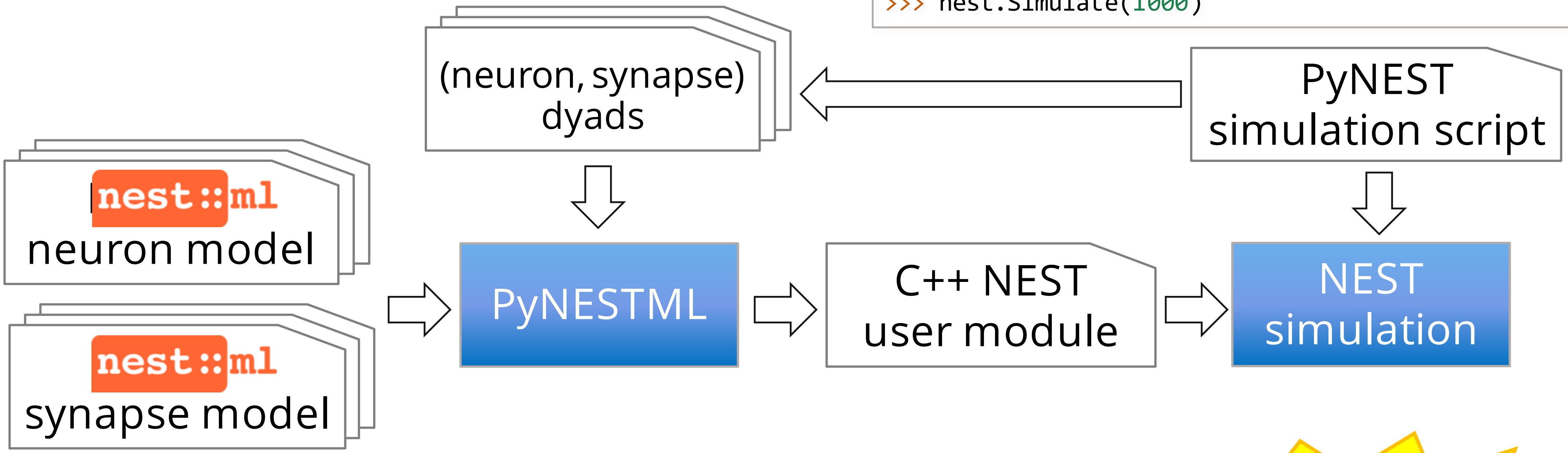
```
>>> nest.Install('nestmlmodule')
>>> pre, post = nest.Create('hodgkin_huxley', 100)
>>> nest.Connect(pre, post,
     'all_to_all', syn_spec={'synapse_model': 'stdp'})
>>> nest.Simulate(1000)
```




```
$ ls -l models
hodgkin_huxley.nestml
stdp.nestml

$ nestml --input_path=models
```

New workflow (with JIT)

```
>>> pre, post = nest.Create('hodgkin_huxley', 100)
>>> nest.Connect(pre, post,
     'all_to_all', syn_spec={'synapse_model': 'stdp'})
>>> nest.Simulate(1000)
```




```
$ ls -l models
hodgkin_huxley.nestml
stdp.nestml
```

```
synapse stdp:
  state:
    w nS = 1 nS
    tr_post real = 0
    tr_pre real = 0
  end

  equations:
    tr_pre' = tr_pre / tau_tr
    tr_post' = tr_post / tau_tr
  end

  input:
    pre_spikes nS <- spike
    post_spikes nS <- post spike
  end

  preReceive:
    w -= alpha * tr_post          # depress synapse
    tr_pre += 1                   # update presynaptic trace
    deliver_spike(w, delay)       # to postsynaptic partner
  end

  postReceive:
    w += alpha * tr_pre           # potentiate synapse
    tr_post += 1                  # update postsynaptic trace
  end

  # parameters: tau_tr, alpha, delay
end
```

```
neuron hodgkin_huxley:
  state:
    V_m mV = -65 mV
    Act_m, Act_n, Inact_h ...
  end

  equations:
    shape syn_psc_kernel = exp(-t / tau_syn)
    inline I_Na pA = ...
    inline I_K pA = ...
    inline I_L pA = g_L * (V_m - E_L)
    V_m' = -(I_Na + I_K + I_L) / C_m
           + convolve(syn_psc_kernel, spikes)
    [...]
  end

  parameters:
    C_m pF = 250 pF
    V_threshold mV = 40 mV
    ...
  end

  update:
    integrate_odes()
    if V_m >= V_threshold:
      emit_spike()
    end
  end
end
```

PyNESTML toolchain:
modular and extensible

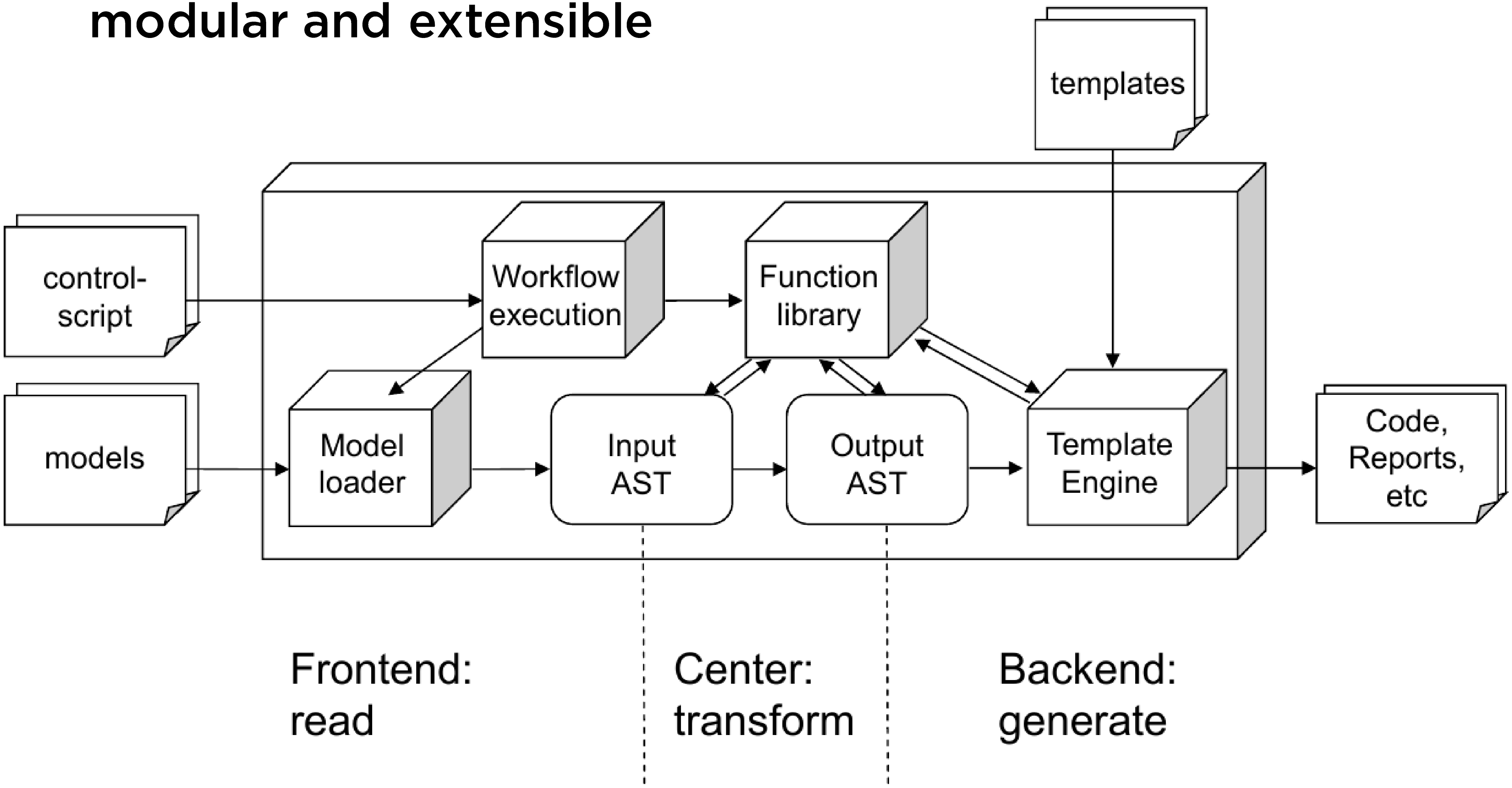
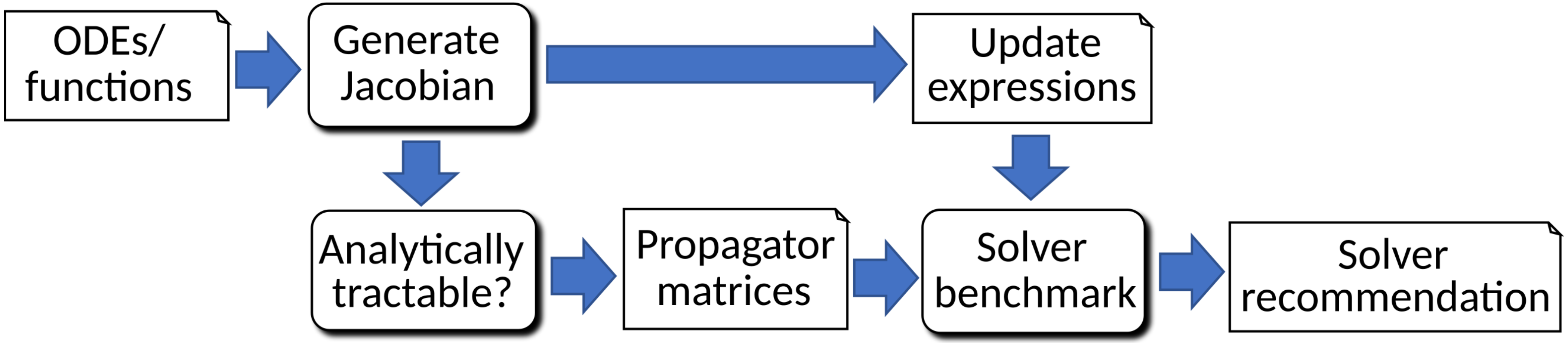


Image credit: Rumpe & Hölldobler 2017

ODE-toolbox:

Automatic selection and generation of integration schemes for systems of ordinary differential equations

- Inputs can be formulated as kernels $f(t) = \ldots$ or as differential equations of any order $d^n f/dt^n = \ldots$
- Symbolic rewriting into system of first-order ODEs
- Propagator matrices for dynamics that admits an analytic solution
- Solver benchmarking and recommendation



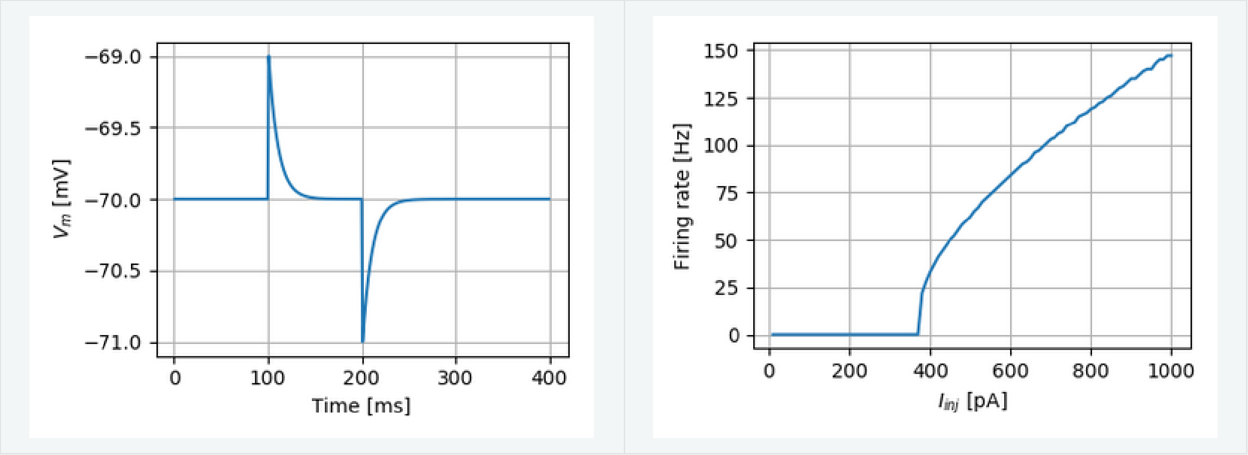
NESTML software development uses best practices in software engineering.

- Unit tests: language feature tests; physical units consistency; etc.
- Integration tests: models are behaviourally validated in one or more simulation runs
- Extensive documentation and automated HTML documentation generation for models: https://nestml.readthedocs.org/
- Open development: https://github.com/nest/nestml
- GNU GPL v2.0 licensed

Models library

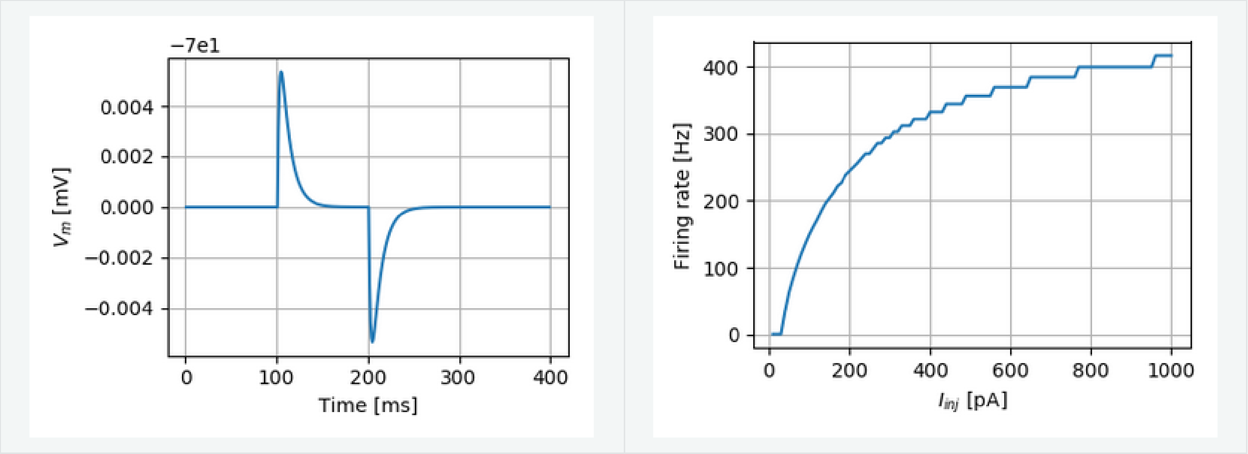
iaf_psc_delta

Source file: iaf_psc_delta.nestml



iaf_psc_exp

Source file: iaf_psc_exp.nestml



NESTML is a **domain-specific modeling language** for the dynamical simulation of point neurons (spiking and rate-based), as well as synapses and synaptic plasticity rules (in alpha version).

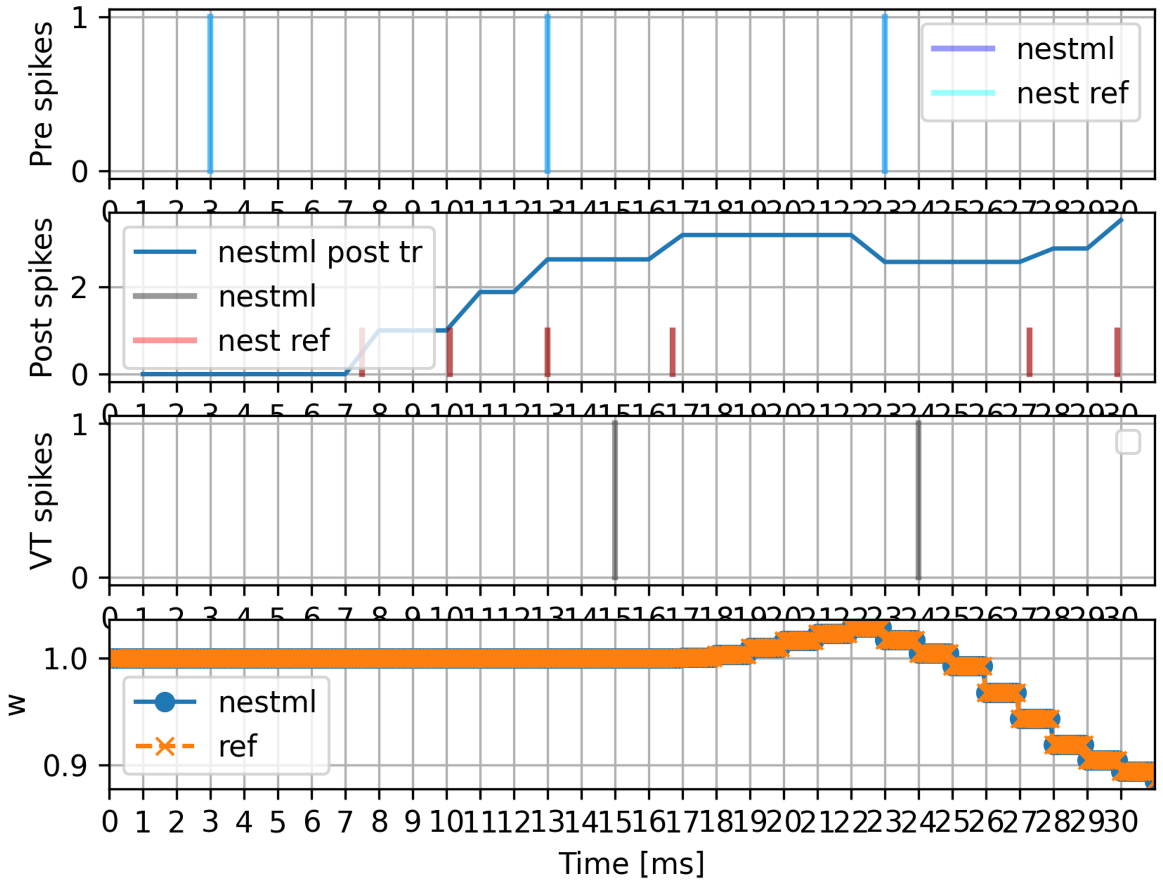
- Low on boilerplate; concise yet expressive syntax
- Direct language support for dynamical equations
- Imperative programming style specification of event handling and generation

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

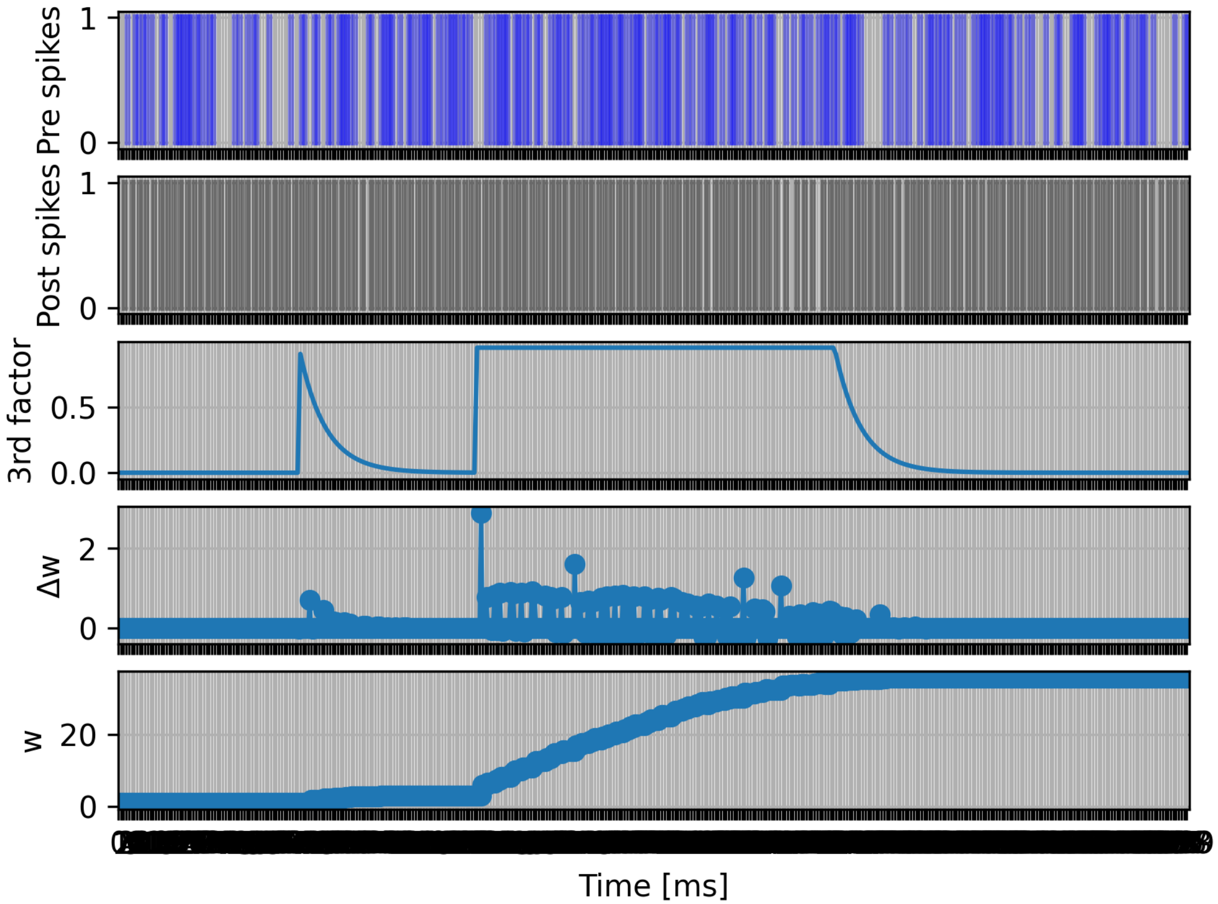
```
neuron iaf_psc_exp:
  state:
    V_m mV = 0 mV
  end

  equations:
    shape G = exp(-t / tau_syn)
    V_m' = -V_abs / tau_m
           + (I_ext + convolve(G, spikes)) / C_m
  end

  parameters:
    C_m      pF = 250 pF
    tau_m    ms = 10 ms
    tau_syn  ms = 2 ms
    V_threshold  mV = 40 mV    # w.r.t. zero!
  end

  input:
    spikes pA <- spike
    I_ext pA <- current
  end

  update:
    integrate_odes()
    if V_abs >= V_threshold:
      V_abs = 0 mV
      emit_spike()
    end
  end
end
```

Neuromodulated STDP




```
synapse stdp_dopa:

  input:
    [...]
    mod_spikes real <- spike
  end

  onReceive(mod_spikes):
    n += 1. / tau_n
  end

  update:              # update from time t to t + resolution()
    # the sequence here matters: the update step for w requires
    # the "old" values of c and n
    w -= c * ( n / tau_s * expm1( -tau_s * resolution() ) \
             - b * tau_c * expm1( -resolution() / tau_c ))
    [...]
    n = n * exp(-resolution() / tau_n)
  end

  [...]
end
```

Postsynaptic dendritic current-modulated STDP




```
synapse stdp_third_factor:
  state:
    w real = .5                                    # assume 0 <= w <= 1
  end

  input:
    [...]
    I_post_dend pA <- continuous
  end

  onReceive(post_spikes):                          # potentiate synapse
    dw real = lambda * pre_trace * ( 1 - w )**mu_plus
    new_w real = w + dw
    I_post_dend = min(I_post_dend, 1 pA)           # clip to 1 pA
    new_w = (I_post_dend / pA) * new_w             # "gating"
            + (1 - I_post_dend / pA) * w           # of the weight update
    w = min(1, new_w)                              # enforce w <= 1
  end

  [...]
end
```

Thank you

Jochen M. Eppler
Abigail Morrison
Markus Diesmann
Konstantin Perun
Pooja Babu

Dimitri Plotnikov
Inga Blundell
Tanguy Fardet
Jessica Mitchell
Sara Konradi

... and to all our users!

Human Brain Project

EBRAINS

JÜLICH
FORSCHUNGSZENTRUM

This software was initially supported by the JARA-HPC Seed Fund *NESTML - A modeling language for spiking neuron and synapse models for NEST* and the Initiative and Networking Fund of the Helmholtz Association and the Helmholtz Portfolio Theme *Simulation and Modeling for the Human Brain*.

This software was developed in part or in whole in the Human Brain Project, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreements No. 720270, No. 785907 and No. 945539 (Human Brain Project SGA1, SGA2 and SGA3).