Introduction to NEST

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 and general features

Built-in neuron models

Built-in synapse models

Custom neuron and synapse models with NESTML

Stimulation devices

Recording devices

Connection management

Parametrization

“Hello world!”

Example: balanced random network

Overview and general features

- **NEST** = **NE**ural **S**imulation **T**echnology
- main focus:

  structure and dynamics of large networks
  of **spiking point neurons**

Overview and general features

- **NEST** = **NE**ural **S**imulation **T**echnology
- main focus:

  structure and dynamics of large networks
  of **spiking point neurons**
- contra indications:

  neuron models with detailed neuronal morphology
  (see **NEURON**)

Overview and general features

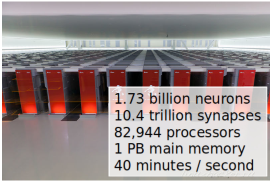
- **NEST** = **NE**ural **S**imulation **T**echnology
- main focus:

  structure and dynamics of large networks
  of **spiking point neurons**

- contra indications:

  neuron models with detailed neuronal morphology
  (see **NEURON**)

- runs on laptops as well as supercomputers (Helias et al. 2012; Jordan et al. 2018)
- supported platforms:
  - Linux
  - Mac OS X
  - Windows via virtual machine



Overview and general features

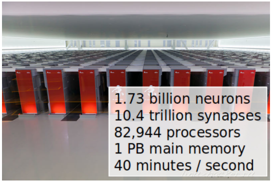
- **NEST** = **NE**ural **S**imulation **T**echnology
- main focus:

  structure and dynamics of large networks
  of **spiking point neurons**

- contra indications:

  neuron models with detailed neuronal morphology
  (see **NEURON**)

- runs on laptops as well as supercomputers (Helias et al. 2012; Jordan et al. 2018)



- supported platforms:
  - Linux
  - Mac OS X
  - Windows via virtual machine
- sources:
  - web: `https://www.nest-simulator.org`
  - code: `https://github.com/nest/nest-simulator`
  - docs: `https://nest-simulator.readthedocs.io`

Overview and general features

- open-source software, licensed under GNU General Public License
  - first release: “SYNOD” (1994)
  - latest release: NEST 3.5 (Feb. 2023)

Overview and general features

- open-source software, licensed under GNU General Public License
  - first release: “SYNOD” (1994)
  - latest release: NEST 3.5 (Feb. 2023)

- C++ kernel

- Python frontend **PyNEST**

Overview and general features

- open-source software, licensed under GNU General Public License
  - first release: “SYNOD” (1994)
  - latest release: NEST 3.5 (Feb. 2023)

- C++ kernel

- Python frontend **PyNEST**

- hybrid parallelization:
  - multi-threading for efficient usage of multi-processor machines
  - MPI-parallelism for computer clusters and super computers

Overview and general features

- open-source software, licensed under GNU General Public License
  - first release: “SYNOD” (1994)
  - latest release: NEST 3.5 (Feb. 2023)
- C++ kernel
- Python frontend **PyNEST**
- hybrid parallelization:
  - multi-threading for efficient usage of multi-processor machines
  - MPI-parallelism for computer clusters and super computers
- large user community (see `https://www.nest-simulator.org/community`)
  - active user mailing list with short response latencies
  - yearly NEST conference (see `https://nest-simulator.org/conference`)

Overview and general features

- open-source software, licensed under GNU General Public License
  - first release: “SYNOD” (1994)
  - latest release: NEST 3.5 (Feb. 2023)
- C++ kernel
- Python frontend **PyNEST**
- hybrid parallelization:
  - multi-threading for efficient usage of multi-processor machines
  - MPI-parallelism for computer clusters and super computers
- large user community (see `https://www.nest-simulator.org/community`)
  - active user mailing list with short response latencies
  - yearly NEST conference (see `https://nest-simulator.org/conference`)
- development
  - driven by scientific needs
  - focus on efficiency, flexibility, accuracy, and reproducibility
  - based on agile, continuous-integration workflows

Overview and general features

- open-source software, licensed under GNU General Public License
  - first release: “SYNOD” (1994)
  - latest release: NEST 3.5 (Feb. 2023)
- C++ kernel
- Python frontend **PyNEST**
- hybrid parallelization:
  - multi-threading for efficient usage of multi-processor machines
  - MPI-parallelism for computer clusters and super computers
- large user community (see `https://www.nest-simulator.org/community`)
  - active user mailing list with short response latencies
  - yearly NEST conference (see `https://nest-simulator.org/conference`)
- development
  - driven by scientific needs
  - focus on efficiency, flexibility, accuracy, and reproducibility
  - based on agile, continuous-integration workflows
- citing NEST: see Zenodo, e.g. (Sinha et al. 2023)

Built-in neuron models

- integrate-and-fire models (`iaf_*`, `*if_*`)
  - with current-based (`iaf_psc_*`)
  - and conducance based synapses (`iaf_cond_*`)
  - and different synaptic kernels (`*_delta`, `*_exp`, `*_alpha`)
- adaptive exponential IaF models (AdEx; `aeif_*`)
- generalized leaky integrate-and-fire models (glif; `gif_*`)
- multi-timescale adaptive threshold model (`mat2_*`,`amat2_*`)
- Izhikevich model (`izhikevich`)
- Hodgkin-Huxley type models (`hh_*`)
- neuron models with multiple (few) compartments (`*_mc_*`)
- firing rate neurons (`tanh_*`, `threshold_lin_*`, `sigmoid_*`, `siegert_*`)
- binary neuron models (`mcculloch_pitts_*`, `ginzburg_*`)
- …and many more

(for an overview of all built-in models, see https://nest-simulator.readthedocs.io/en/stable/models)

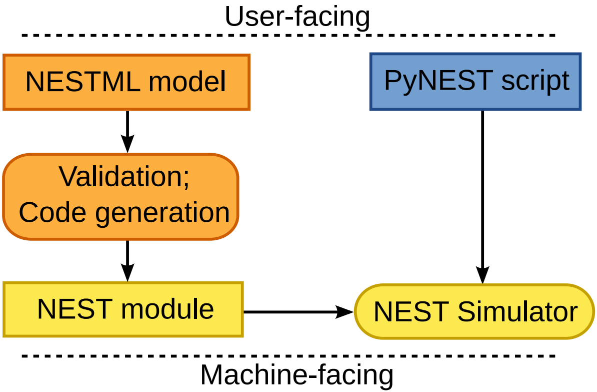
Built-in synapse models

- current-based and conductance-based synapses (see neuron models)
- gap junctions (`gap_junction`)
- static synapses (`static_synapse*`)
- various types of synaptic plasticity
  - various forms of short term plasticity (`tsodyks*`, `quantal_stp_*`)
  - various forms of spike-timing-dependent plasticity (`stdp_*`)
  - STDP plus third factors (`clopath_*`, `stdp_dopamine_*`)
  - structural plasticity (Butz and Ooyen 2013)
- stochastic synapses (synaptic failure; `bernoulli_synapse`)
- …and many more

(for an overview of all built-in models, see `https://nest-simulator.readthedocs.io/en/stable/models`)

Custom neuron and synapse models with NESTML

- domain-specific language for neuron and synapse models



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

Stimulation devices

Spike generators

- spikes at prescribed points in time: `spike_generator`
- realizations of homogeneous or inhomogeneous Poisson point processes:
  - `poisson_generator`
  - `inhomogeneous_poisson_generator`
  - `sinusoidal_poisson_generator`
- realizations of homogeneous or inhomogeneous Gamma point processes:
  - `gamma_sup_generator`
  - `sinusoidal_gamma_generator`

Current generators

- constant current: `dc_generator`
- sinusoidal current: `ac_generator`
- step-wise constant current: `step_current_generator`
- noisy current: `noise_generator`

(see https://nest-simulator.readthedocs.io/en/stable/devices/stimulate_the_network.html)

Recording devices

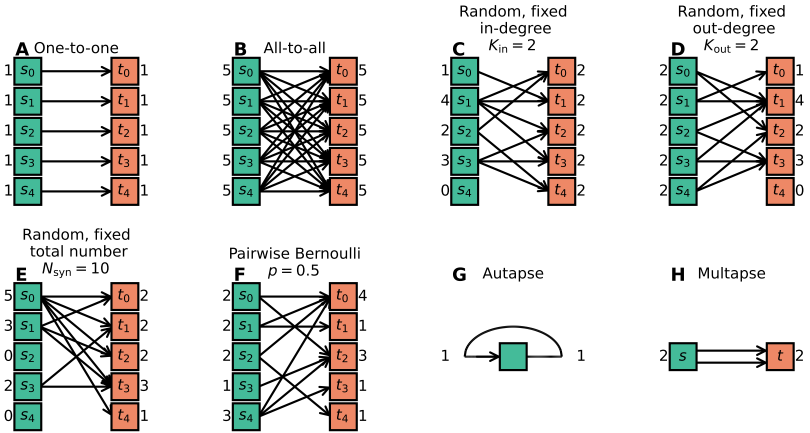
- spikes : `spike_recorder`
- analog quantities (membrane potentials, conductances, . . .): `multimeter`
- synaptic weights: `weight_recorder`

(see https://nest-simulator.readthedocs.io/en/stable/devices/record_from_simulations.html)

Connection management

- large repertoire of efficient built-in connection routines (Ippen et al. 2017), incl.
  - various forms of deterministic and random connectivity patterns
  - spatially structured networks

(see `https://nest-simulator.readthedocs.io/en/stable/synapses/connection_management.html`)



(Senk et al. 2022)

Parametrization

- parameters: specification of, e.g.,
  - initial conditions
  - neuron or device properties
  - positions in physical space
  - connection probabilities, synaptic weights, delays

(see `https://nest-simulator.readthedocs.io/en/stable/neurons/parametrization.html`)

Parametrization

- parameters: specification of, e.g.,
  - initial conditions
  - neuron or device properties
  - positions in physical space
  - connection probabilities, synaptic weights, delays
- parameter types:
  - random parameters (built-in RNGs)

    (see https://nest-simulator.readthedocs.io/en/stable/nest_behavior/random_numbers.html)
  - spatial parameters
  - spatially distributed parameters
  - mathematical functions
  - clipping, redrawing, and conditional parameters
  - combination of parameters

(see https://nest-simulator.readthedocs.io/en/stable/neurons/parametrization.html)

“Hello world!”

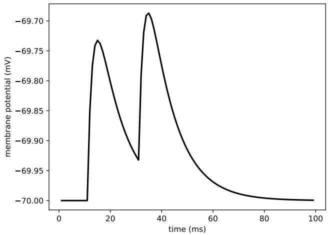
```
import nest     # import NEST module
import matplotlib.pyplot as plt # for plotting

nest.ResetKernel() # reset simulation kernel

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

# create a spike generator, and set it up to create two spikes at 10 and 30ms
spikegenerator=nest.Create('spike_generator', params={'spike_times': [10.,30.]})

# create multimeter and set it up to record the membrane potential V_m
multimeter=nest.Create('multimeter', {'record_from': ['V_m']})

# connect spike generator with neuron with synaptic weight 100 pA
nest.Connect(spikegenerator, neuron,syn_spec={'weight': 50.0})

nest.Connect(multimeter, neuron)  # connect multimeter to the neuron

nest.Simulate(100.) # run simulation for 100ms

# read out recording time and voltage from voltmeter
times=multimeter.get('events')['times']
voltage=multimeter.get('events')['V_m']

# plot results
plt.figure(1)
plt.clf()
plt.plot(times,voltage,'k-',lw=2)
plt.xlabel('time (ms)')
plt.ylabel('membrane potential (mV)')
plt.savefig('./figures/hello_world.pdf')
```

(see `hello_world.py`)

“Hello world!”

```python
import nest    # import NEST module
import matplotlib.pyplot as plt # for plotting

nest.ResetKernel() # reset simulation kernel

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

# create a spike generator, and set it up to create two spikes at 10 and 30ms
spikegenerator=nest.Create('spike_generator', params={'spike_times': [10.,30.]})

# create multimeter and set it up to record the membrane potential V_m
multimeter=nest.Create('multimeter', {'record_from': ['V_m']})

# connect spike generator with neuron with synaptic weight 100 pA
nest.Connect(spikegenerator, neuron,syn_spec={'weight': 50.0})

nest.Connect(multimeter, neuron)  # connect multimeter to the neuron

nest.Simulate(100.) # run simulation for 100ms

# read out recording time and voltage from voltmeter
times=multimeter.get('events')['times']
voltage=multimeter.get('events')['V_m']

# plot results
plt.figure(1)
plt.clf()
plt.plot(times,voltage,'k-',lw=2)
plt.xlabel('time (ms)')
plt.ylabel('membrane potential (mV)')
plt.savefig('./figures/hello_world.pdf')
```

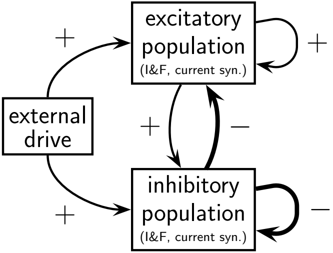
hello_world.pdf:



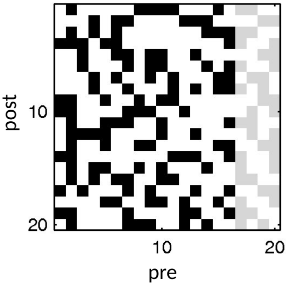
(see `hello_world.py`)

Example: balanced random network

simple model of a local cortex volume (Brunel 2000)
($\sim 1\text{mm}^3$, $\sim 10^{4\ldots5}$ neurons)



Connectivity matrix $J = \{J_{ij}\}$

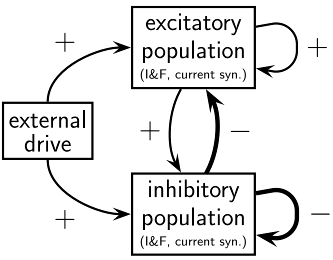


- N_E excitatory, N_I inhibitory neurons, $N = N_\mathsf{E} + N_\mathsf{I} \sim 10^{4\ldots5}$
- sparse, random connectivity with fixed in-degree $K = N/10$
- LIF dynamics with current-based synapses with exponential shape:

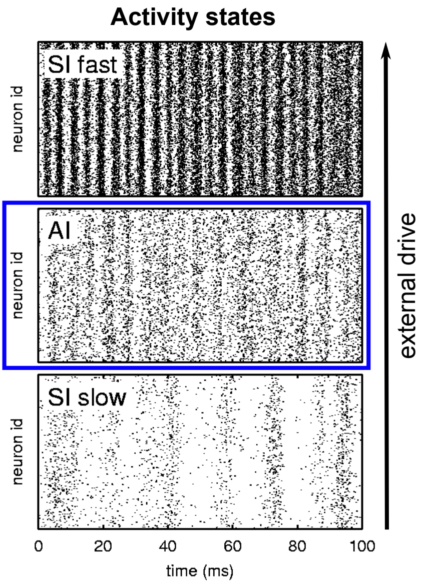
$$\begin{aligned} \tau_\mathsf{m}\dot{V}_i &= -V_i(t) + R\left(I_i^{\mathsf{net}}(t) + I^{\mathsf{ext}}\right) \quad (i \in \{1,\ldots,N\}) \\ I_i^{\mathsf{net}}(t) &= \sum_j J_{ij}(h * s_j)(t-d) \quad \text{with} \quad h(t) = e^{-t/\tau_\mathsf{s}}\Theta(t) \end{aligned}$$

Example: balanced random network

simple model of a local cortex volume (Brunel 2000)
($\sim 1\mathrm{mm}^3$, $\sim 10^{4\dots5}$ neurons)

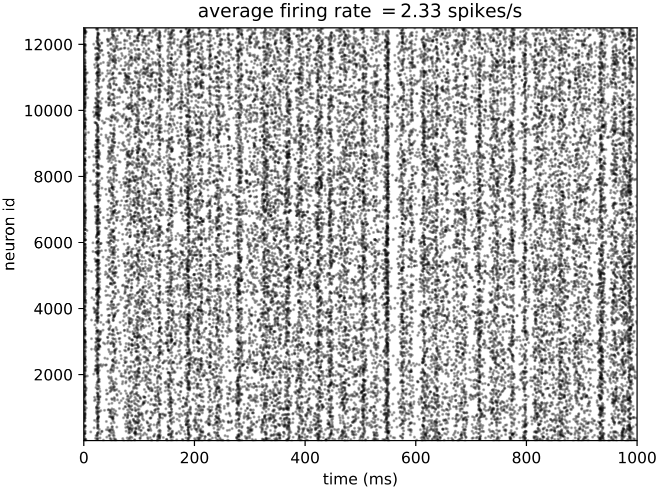


- in-vivo like activity
  - large membrane potential fluctuations
  - low firing rates
  - irregular spiking
- global oscillatory modes in various frequency bands



Example: balanced random network

implementation: see `balanced_random_network.py`



Acknowledgments

This presentation is based on previous work by many people, in particular:

- Hannah Bos
- David Dahmen
- Moritz Deger
- Jochen Martin Eppler
- Espen Hagen
- Charl Linssen
- Abigail Morrison
- Jannis Schuecker
- Johanna Senk
- Tom Tetzlaff
- Sacha van Albada
- Alexander van Meegen

Thanks

References I

Brunel, Nicolas (2000). “Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons”. In: **Journal of Computional Neuroscience** 8.3, pp. 183–208. DOI: `10.1023/a:1008925309027`.

Butz, Markus and Arjen van Ooyen (2013). “A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions”. In: **PLoS Computational Biology** 9.10, e1003259. DOI: `10.1371/journal.pcbi.1003259`.

Helias, Moritz et al. (2012). “Supercomputers ready for use as discovery machines for neuroscience”. In: **Frontiers in Neuroinformatics** 6, p. 26. DOI: `10.3389/fninf.2012.00026`.

Ippen, Tammo et al. (2017). “Constructing neuronal network models in massively parallel environments”. In: **Frontiers in Neuroinformatics** 11, p. 30. DOI: `10.3389/fninf.2017.00030`.

Jordan, Jakob et al. (2018). “Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers”. In: **Frontiers in Neuroinformatics** 12, p. 2. DOI: `10.3389/fninf.2018.00002`.

Senk, Johanna et al. (2022). “Connectivity Concepts in Neuronal Network Modeling”. In: **PLoS Computational Biology** 18.9, e1010086. DOI: `10.1371/journal.pcbi.1010086`.

Sinha, Ankur et al. (2023). **NEST 3.4**. DOI: `10.5281/ZENODO.6867799`. URL: `https://zenodo.org/record/6867799`.