Convenient simulation of spiking networks with NEST2

Jochen Martin Eppler[1,2], Moritz Helias[2], Eilif Muller[3], Markus Diesmann[2,4], Marc-Oliver Gewaltig[1]

eppler@biologie.uni-freiburg.de, helias@bccn.uni-freiburg.de, eilif.mueller@epfl.ch, diesmann@brain.riken.jp, marc-oliver.gewaltig@honda-ri.de

1 Honda Research Institute Europe
Carl-Legien-Str. 30
63073 Offenbach/Main, Germany
http://www.honda-ri.de

2 Bernstein Center for Computational Neuroscience
Hansastraße 9a
79104 Freiburg, Germany
http://www.bccn.uni-freiburg.de

3 Laboratory of Computational Neuroscience
Swiss Federal Institute of Technology, EPFL
1015 Lausanne, Switzerland
http://lcn.epfl.ch

4 RIKEN Brain Science Institute
Wako City
351-0198 Saitama, Japan
http://www.brain.riken.jp

Overview

- NEST is a simulator for neural systems at the level of spiking neurons.
- NEST is optimized for large networks
- It runs efficiently on single- and multi-processor computers and clusters.
- Simulations can be written in Python.
- This poster illustrates the main features of NEST from the user's perspective.

What you can do with NEST

NEST is a network simulator for models that focus on the dynamics, size, and structure of neural systems rather than on the exact geometry of individual neurons [1,2].

Examples of such models are:

- Models of sensory processing e.g. in the visual or auditory cortex of mammals [4].
- Models of network activity dynamics, e.g. in laminar cortical networks or balanced random networks.
- Models of spike-synchronization in feed-forward networks like synfire chains [5].
- Learning and plasticity in models of sensory processing.

Modeling paradigm

A simulation with NEST is like an electrophysiological experiment that takes place inside the computer.

- You start with a system to investigate and an idea of what you want to learn from the experiment.
- You build the neural system by arranging model neurons and connecting them according to your model.
- You can use different model neurons in one network.
- You define layers, areas, and sub-networks.
- You can use different types of synapses (e.g. STDP, facilitation, and depression), to connect your model neurons.
- Special elements, called devices, are responsible for measurement and stimulation.
- After the network is set up, the simulation kernel executes the model for a period of simulated time.

Models

A network consists of nodes and their connections. Nodes are neurons, devices, and sub-networks. Connections are the channels along which different types of events like spikes or currents can travel.

NEST offers different models for neurons, synapses, and devices:

- Integrate and fire models with current-, and conductance-based synapses.
- Neuron models with precise spike-time integration [6].
- Hodgkin-Huxley models with current- and conductance based synapses.
- Synapses with spike-time dependent plasticity (STDP) or short-term facilitation and depression.
- Stimulus devices, for different currents, Poisson and custom spike-trains.
- Devices to record spikes and membrane potentials.

Some of NEST's neuron models

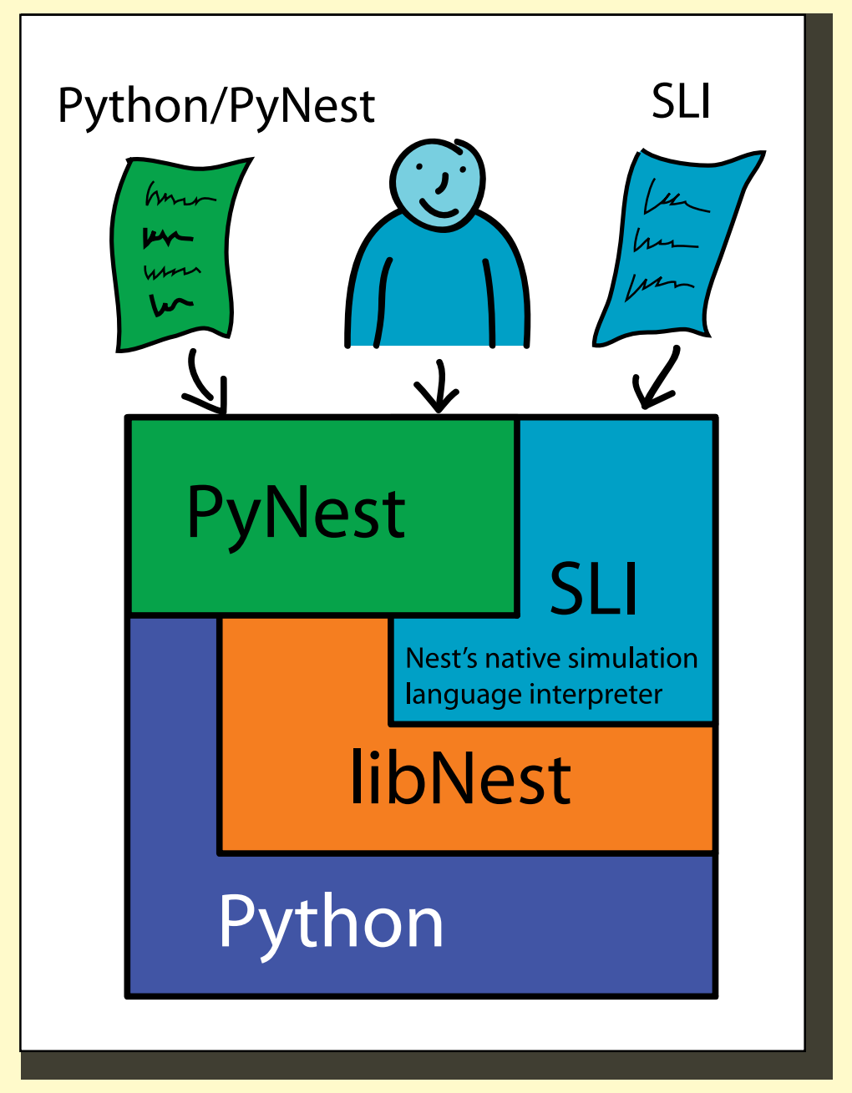
Model name	Model type	Synapse response
iaf_neuron	integrate and fire	alpha function current
iaf_psc_delta	integrate and fire	delta function current
iaf_psc_exp	integrate and fire	exponential current
iaf_psc_alpha	integrate and fire	alpha function current
iaf_cond_exp	integrate and fire	exponential conductance
iaf_cond_alpha	integrate and fire	alpha function conductance
hh_psc_alpha	Hodgkin-Huxley	alpha function current
hh_cond_exp_traub	Hodgkin-Huxley	exponential conductance

Models for neurons, devices, synapses, and events are implemented in C++. You can add your own models to NEST.

NEST and Python

There are two ways to use NEST:

1. As Python module (PyNEST [3]).
2. With NEST's native simulation language SLI.



Python is a high-level language with a simple syntax that is easy to read and easy to learn [3].

It provides many powerful modules for scientific computing and visualization.

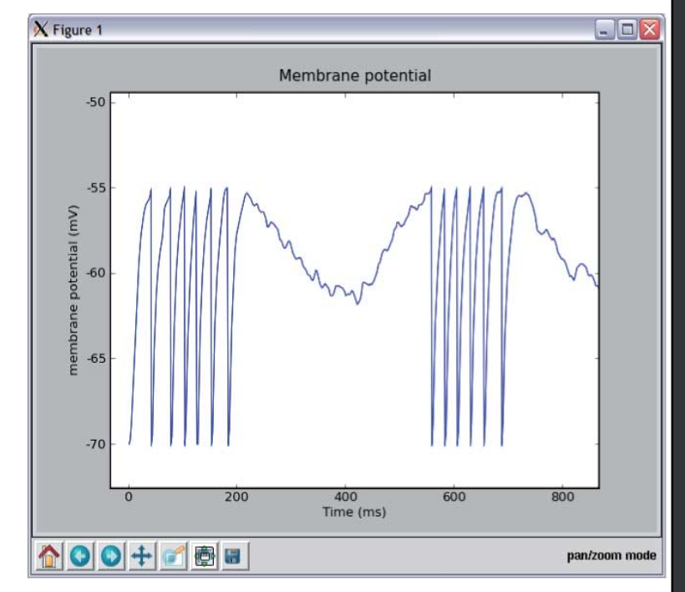
An example

We simulate a single integrate and fire neuron that receives a sine current as well as excitatory and inhibitory Poisson noise as input. The neuron is simulated for 1000 ms.

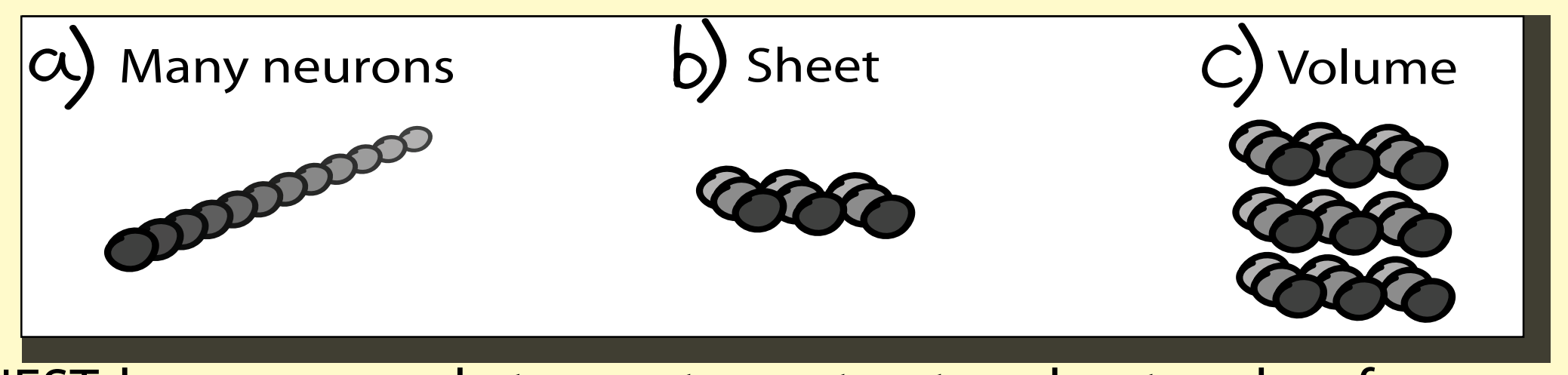
The panel shows the simulation script and graph of the membrane potential of the neuron, using Python's Matplotlib [9]

One Integrate and Fire Neuron

```
#! /usr/bin/env python

import nest
import nest.voltage_trace

neuron = nest.Create("iaf_neuron")

noise = nest.Create("poisson_generator", 2)
nest.SetStatus(noise, [{"rate": 80000.0},
                       {"rate": 20000.0}])

sine = nest.Create("ac_generator")
nest.SetStatus(sine, [{"amplitude": 100.0,
                       "frequency": 2.0}])

voltmeter = nest.Create("voltmeter")
nest.SetStatus(voltmeter, {"withgid": True, "withtime": True})

nest.ConvergentConnect(noise, neuron, [1.0, -1.0], 1.0)
nest.Connect(voltmeter, neuron)
nest.Connect(sine, neuron)

nest.Simulate(1000.0)
nest.voltage_trace.from_device(voltmeter)
```



Creating large networks



NEST has commands to create unstructured networks of many neurons, or structured networks where the neurons are arranged in sheets or volumes.

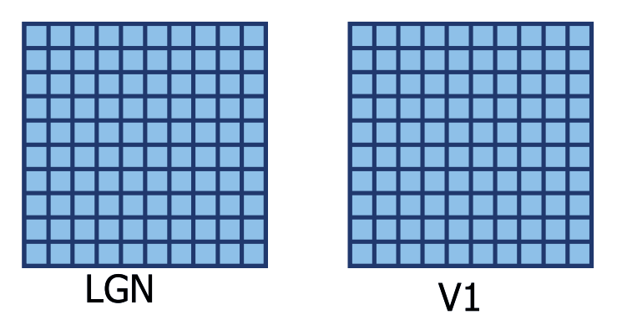
`Create("iaf_neuron",n)` creates **n** neurons and returns a list with IDs of the new nodes.

`LayoutNetwork("iaf_neuron",[2,10,10])` creates multi-dimensional networks, here two 10 x 10 sheets of integrate and fire neurons. A list of arbitrary length defines the shape of the network. Sheets are represented as nested sub-networks.

`PrintNetwork()` shows a part of the network. Optional parameters determine how much of the network is shown.

Layered Networks

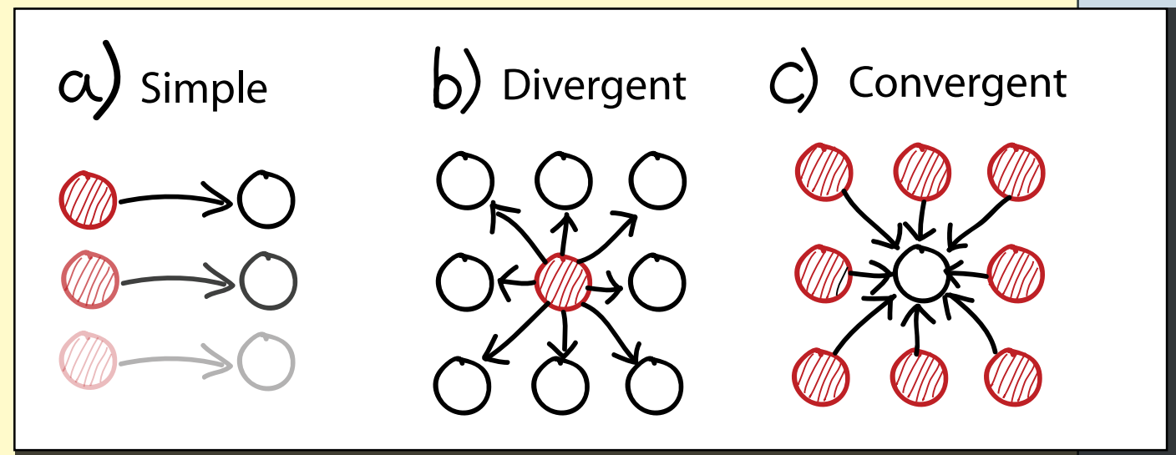
```
In [2]:nest.LayoutNetwork("iaf_neuron",[2,10,10])
Out[2]:[1]

In [3]:nest.PrintNetwork()
+-[0] root dim=[1 2 10 10]
   |
   +-[1] subnet dim=[2 10 10]

In [4]:nest.PrintNetwork(2)
+-[0] root dim=[1 2 10 10]
   |
   +-[1] subnet dim=[2 10 10]
      |
      +-[1] subnet dim=[10 10]
      +-[2] subnet dim=[10 10]
```



- Two visual areas, LGN and V1
- Each area represented by a 2 dimensional sheet of 10 by 10 integrate and fire neurons.

Connecting neurons

All of NEST's connection commands base on three primitives: a) Simple connections between two neurons, (b) divergent connections from one neuron to a set of targets, and (c) convergent connections from many neurons to a common target.



The example below shows how the random connections of the Brunel model [8] are drawn with these primitives.

Connectivity for the random network described by Brunel (2000)

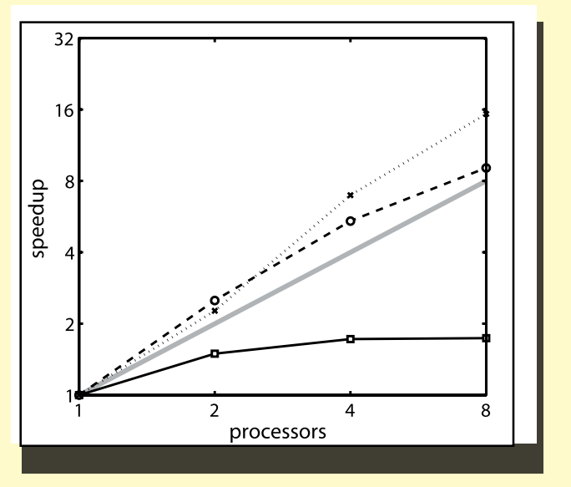
```
# Creating synapse models
nest.CopyModel("static_synapse","excitatory",{"weight":J_ex, "delay":delay})
nest.CopyModel("static_synapse","inhibitory",{"weight":J_in, "delay":delay})

print "Connecting devices."
nest.DivergentConnect(noise,nodes_ex,model="excitatory")
nest.DivergentConnect(noise,nodes_in,model="excitatory")

nest.ConvergentConnect(range(1,N_rec+1),espikes,model="excitatory")
nest.ConvergentConnect(range(NE+1,NE+1+N_rec),ispikes,model="excitatory")

print "Connecting network."
nest.RandomConvergentConnect(nodes_ex, nodes_ex+nodes_in, CE,model="excitatory")
nest.RandomConvergentConnect(nodes_in, nodes_ex+nodes_in, CI,model="inhibitory")
```

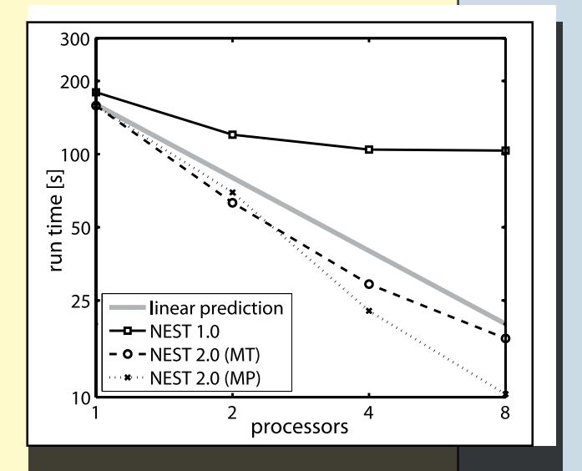
Parallel simulation

NEST supports parallel and distributed simulation.

On single computers, NEST distributes the network over the available processors, each executing one part. This reduces the simulation time considerably [6].

On computer clusters, NEST distributes the network over the available computers. Each computer creates its own part of the network and stores only connections to its own neurons. This reduces the run-time of simulations and increases the computer memory that is available to NEST [7].





Interested in NEST?

NEST is developed by the Neural Simulation Technology Initiative (NEST Initiative), which was founded in 2001 with the aim to advance and document methods for the simulation of large neural systems.

The NEST Initiative regularly prepares releases of the NEST software.

How to get NEST

This poster describes the forthcoming version 2.0 of NEST. Beta-versions are available free of charge at:

`http://www.nest-initiative.org.`

To compile NEST, you need a recent C++ compiler (e.g. GCC) and a POSIX compatible operating system.

NEST runs on Linux, Solaris, Tru64, MacOS, BlueGene, and others.

Acknowledgements

We would like to thank all members of the NEST Initiative and in particular (in alphabetical order): Kittel Austvoll, Jochen Martin Eppler, Moritz Helias, Rüdiger Kupper, Wiebke Potjans, and Sven Schrader.

Partially funded by DAAD 313-PPP-N4-lk, DIP F1.2, EU Grant 15879 (FACETS), and BMBF Grant 01GQ0420 to the Bernstein Center for Computational Neuroscience Freiburg.

References

1. Gewaltig M-O, Diesmann M. NEST (Neural Simulation Tool). Scholarpedia p 11204. Online at www.scholarpedia.org. 2007
2. Diesmann M and Gewaltig M-O. NEST: An Environment for Neural Systems Simulations. Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001.Ges. für Wiss. Datenverarbeitung.2002
3. Eppler J M, Helias M, Muller E, Diesmann M, Gewaltig M-O. PyNEST: A convenient interface to the NEST simulator. Frontiers in Neuroinformatics 2009 2:12. doi:10.3389/neuro.11.012.2008.
4. Gewaltig M-O, Körner U, and Körner E. A model of surface detection and orientation tuning in primate visual cortex. Computational Neuroscience: Trends in Research 2002. Elsevier
5. Diesmann M, Gewaltig M-O, and Aertsen A. Stable propagation of synchronous spiking in cortical neural networks. Nature.1999.
6. Morrison A, Straube S, Plesser H E, and Diesmann, M. Exact subthreshold integration with continuous spike times in discrete time neural network simulations. 2007 Neural Computation 19(1):47-79
7. Plesser H E, Eppler J M, Morrison A, Diesmann M, Gewaltig M-O. Efficient Parallel Simulation of Large-Scale Neuronal Networks on Clusters of Multiprocessor Computers. Lecture Notes in Computer Science Volume 4641/2007, 672-681
8. Brunel N. Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons. 2000 J. Computational Neuroscience 8, 183-208
9. Matplotlib, `http://matplotlib.sf.net`