

# A common language for neuronal networks in software and hardware

Andrew Davison, Eilif Muller, Daniel Brüderle, and Jens Kremkow

*A new interface allows simulations to be run without modification in multiple software simulators and neuromorphic hardware, greatly simplifying translation and cross-validation of models.*

Fast information processing in the brain is performed by the exchange of brief electrical pulses (called action potentials, or spikes) between neurons that are connected in complex networks. Models of such networks may either be simulated in software<sup>1</sup> or implemented in VLSI hardware.<sup>2</sup> Numerical simulation in software has the advantages of flexibility and the commodity status of digital computers, with simulations being performed on systems ranging from individual desktop machines, through clusters, up to supercomputers.<sup>3</sup> The main advantages of physical emulation in analog or mixed-signal neuromorphic hardware arise from the locally analog and massively parallel nature of the computations together with the very small spatial scale of VLSI components. The benefits are high scalability, computational speed, and low power consumption.

Typically, specification of neuronal network models—the mechanisms of individual neurons and the pattern and properties of the connections between them—is done by editing a configuration file, using a graphical interface to set parameters, or, most flexibly, writing a script in an interpreted programming language. A major problem that impedes communication between researchers, reproducibility of results, and building upon previous work, is that each software simulator or hardware system has its own configuration format, graphical interface or special-purpose programming language. This makes the translation of a model from one system to another—and especially from software to hardware—arduous, time-consuming and error-prone.

However, the availability of multiple simulators and hardware systems has many advantages, particularly in cross-checking results and in exploiting the complementary features

```
from pyNN.neuron import *
from pyNN.random import RandomDistribution
cell_params = {
    'tau_m': 20.0, 'tau_syn_E': 5.0,
    'cm': 0.2, 'tau_syn_I': 10.0,
    'v_rest': -49.0, 'v_reset': -60.0,
    'v_thresh': -50.0, 'tau_refrac': 5.0
}
pE = Population(4000, IF_cond_exp, cell_params)
pI = Population(1000, IF_cond_exp, cell_params)
unif_distr = RandomDistribution('uniform', [-50,-70])
pE.randomInit(unif_distr)
pI.randomInit(unif_distr)
FPC = FixedProbabilityConnector
exc_conn = FPC(0.02, weights=0.004, delays=0.1)
inh_conn = FPC(0.02, weights=0.051, delays=0.1)
e2e = Projection(pE, pI, exc_conn, target='excitatory')
e2i = Projection(pE, pI, exc_conn, target='excitatory')
i2e = Projection(pI, pE, inh_conn, target='inhibitory')
i2i = Projection(pI, pI, inh_conn, target='inhibitory')
pE.record(1000)
pI.record()
pE.record_v([pE[0], pE[1]])
run(1000.0)
pE.printSpikes("pE.spk")
pI.printSpikes("pI.spk")
pE.print_v("pE.v")
end()
```

*Figure 1. Example code for a balanced network of integrate-and-fire neurons. To run the simulation with the NEST simulator instead of with NEURON, we simply change the first line to `from pyNN.nest import *`.*

of different systems, e.g., the flexibility of a simulator and the speed of a hardware system. A common interface for specifying models that would work across multiple systems would retain the benefits of simulator/hardware diversity while reducing or removing the translation barrier. We have developed such a common interface, PyNN (pronounced 'pine') that enables simulation scripts to be run on any supported system (currently four different software simulators and one neuromorphic hardware system) *without modification*.

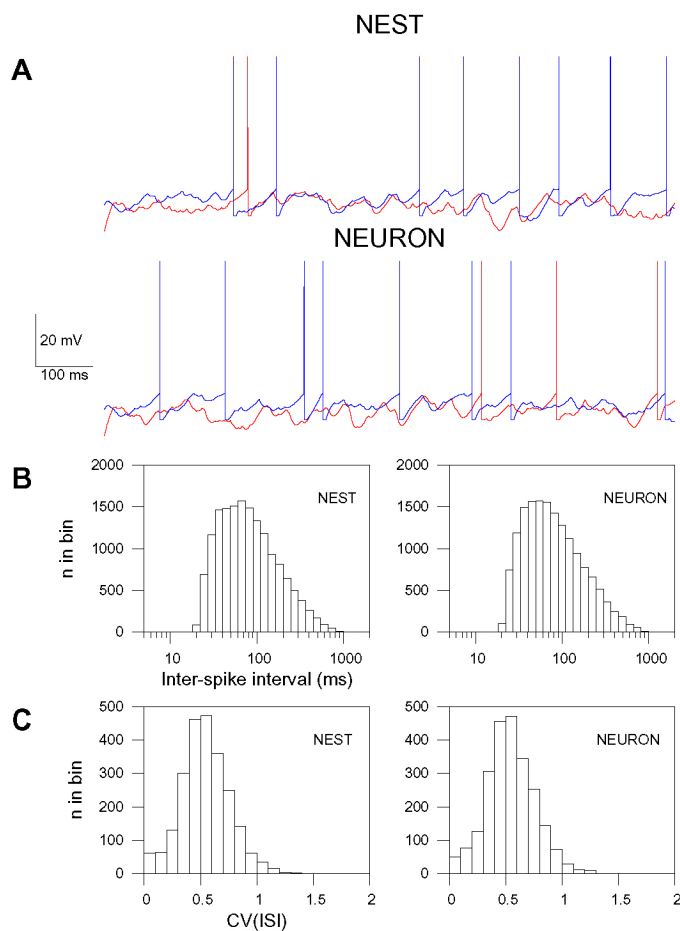
*Continued on next page*

PyNN<sup>4</sup> (available from <http://neuralensemble.org/PyNN>), is both an application programming interface (API) for the Python programming language<sup>5</sup> and an implementation of the interface for a number of systems (currently the NEURON,<sup>6</sup> NEST,<sup>7</sup> PCSIM<sup>8</sup> and Brian<sup>9</sup> simulators and the FACETS neuromorphic mixed-signal VLSI system).<sup>10,11</sup> A simulation script is written in Python, with neuronal network modelling functions and classes provided by PyNN, and optionally exploiting the extensive capabilities for numerical data analysis and visualisation available in Python or in third-party extensions such as NumPy.<sup>12</sup> It can then be run on any supported ‘back-end’ system without modification, except for a single parameter specifying which back-end to use.

PyNN takes care of translating the neuron, synapse and network models into the required form for a given simulator, consistent handling of physical units, and consistent handling of random numbers, and provides a high-level, object-oriented interface to enable structured development of large-scale, complex models. A simple example of a network of excitatory and inhibitory neurons exhibiting self-sustained activity is shown in Figures 1 and 2.

Adding support for a new simulator or hardware back-end requires writing a Python module that implements the PyNN API for that system. However, much of the code in the current implementation is shared between back-ends, and only a small number of simulator-specific functions and classes need be defined to take advantage of the entire API and future extensions to it. It is not required that the simulator or control software have a Python interface, since code-generation can be used in this case, but obviously this makes two-way communication and interactive use more difficult. The API is defined independently of any particular implementation, and is versioned. This means that developers of an interface to a new system are not required to contribute code back to the project (although this is welcomed), but simply need to implement the published API in order to take advantage of being able to run a PyNN simulation on their system.

In summary, PyNN greatly simplifies the tasks of writing simulator-independent neuronal network models, of transferring models from software simulators to neuromorphic hardware systems, and of cross-validating simulation results between different systems. In the future, we hope to see implementations of the interface for further simulators and hardware systems (developers interested in an implementation for their own system are encouraged to contact us) and to implement inter-operability with other tools for neuroscience model description, such as NeuroML.<sup>13</sup> We expect that increased adoption of PyNN will



**Figure 2.** Results of running the code example given in Figure 1, with NEURON and NEST as back-end simulators. (A) Membrane potential traces for two excitatory neurons. (B) Distribution of pooled inter-spike intervals (ISIs). (C) Distribution over neurons of the coefficient of variation of the ISI (CV(ISI)). This figure illustrates the limitations of reproducibility between simulators. Even though the specification of the model is identical in every detail between the two simulators, the network is very sensitive to small perturbations due to floating point round-off, etc., which lead to a divergence of the traces of individual neurons after about 50ms (A), although the statistical properties of the networks (B,C) are practically identical. For hardware systems, with intrinsic component variability and noise, exact reproducibility is even further from being achievable.

accelerate progress in computational neuroscience by facilitating reproducibility and cross-validation of published models and by stimulating component reuse, and that it will close the gap

Continued on next page

between computational neuroscientists and neuromorphic engineers, making neuromorphic hardware more easily exploitable by non-hardware experts.

*This work was supported by the European Union (FACETS project, FP6-2004-IST-FETPI-015879). Many thanks to all those who have contributed to PyNN, and to the members of the FACETS project for their support.*

## Author Information

### Andrew Davison

UNIC

CNRS

Gif sur Yvette, France

<http://www.davison.webfactional.com/>

<http://www.unic.cnrs-gif.fr/>

Dr Andrew Davison is a research scientist with the CNRS, with research interests in large-scale, data-constrained modelling of neural systems.

### Eilif Muller

Laboratory of Computational Neuroscience

Ecole Polytechnique Fédérale de Lausanne

Lausanne, Switzerland

### Daniel Brüderle

Kirchhoff Institute for Physics

University of Heidelberg

Heidelberg, Germany

### Jens Kremkow

Neurobiology and Biophysics

Albert-Ludwigs-University

Freiburg, Germany

## References

1. R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, J. PH Goodman, F. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, S. E. Boustani, and A. Destexhe, *Simulation of networks of spiking neurons: A review of tools and strategies*, **J. Comput. Neurosci.** **23**, pp. 349–98, 2007. doi:10.1007/s10827-007-0038-6
2. C. A. Mead, **Analog VLSI and Neural Systems**, Addison Wesley, Reading, MA, USA, 1989.
3. H. Markram, *The Blue Brain Project*, **Nat. Rev. Neurosci.** **7**, pp. 153–160, 2006. doi:10.1038/nrn1848
4. A. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, *PyNN: a common interface for neuronal network simulators*, **Front. Neuroinf.** **2:11**, 2009. doi:10.3389/neuro.11.011.2008
5. <http://www.python.org>. Python is a dynamic object-oriented programming language that can be used for many kinds of software development. Accessed 21st May 2009
6. N. T. Carnevale and M. L. Hines, **The NEURON Book**, Cambridge University Press, 2006.
7. M.-O. Gewaltig and M. Diesmann, *NEST (NEural Simulation Tool)*, **Scholarpedia** **2** (4), p. 1430, 2007.
8. D. Pecevski, T. Natschläger, and K. Schuch, *PCSIM: a parallel simulation environment for neural circuits fully integrated with Python*, **Front. Neuroinf.** **3:11**, 2009. doi:10.3389/neuro.11.011.2009
9. D. Goodman and R. Brette, *Brian: a simulator for spiking neural networks in Python*, **Frontiers Neuroinf.** **2:5**, 2008. doi:10.3389/neuro.11.005.2008
10. J. Schemmel, D. Brüderle, K. Meier, and B. Ostendorf, *Modeling Synaptic Plasticity within Networks of Highly Accelerated I&F Neurons*, **Proc. 2007 IEEE Int'l Symp. Circuits and Systems (ISCAS'07)**, IEEE Press, 2007.
11. D. Brüderle, E. Muller, A. Davison, M. E. J. Schemmel, and K. Meier, *Establishing a novel modeling tool: A Python-based interface for a neuromorphic hardware system*, **Front. Neuroinf.** **3:17**, 2009. doi:10.3389/neuro.11.017.2009
12. <http://numpy.scipy.org/>. NumPy is the fundamental package needed for scientific computing with Python. Accessed 21st May 2009
13. P. Gleeson, S. Crook, V. Steuber, and R. Silver, *Using NeuroML and neuroConstruct to build neuronal network models for multiple simulators*, **BMC Neuroscience** **8**, p. P101, 2007. doi:10.1186/1471-2202-8-S2-P1