FACETS

FP6-2004-IST-FETPI 15879
Fast Analog Computing with Emergent Transient States

Milestone M8-4

Next public release of PyNN/FacetsML

Report Version: 1.0

Report Preparation: Andrew Davison (CNRS-UNIC)

Classification: Public

Contract Start Date: 01/09/2005 Duration: 4 Years

Project Co-ordinator: Karlheinz Meier (Heidelberg)

Partners: U Bordeaux, CNRS (Gif-sur-Yvette, Marseille), U Debrecen, TU Dresden, U Freiburg, TU Graz, U Heidelberg, EPFL Lausanne, Funetics S.a.r.l., U London, U Plymouth, INRIA, KTH Stockholm

Project funded by the European Community

under the "Information Society Technologies" Programme

DELIVERABLE SUMMARY SHEET

Project Number: FP6-2004-IST-FETPI 15879

Project Acronym: FACETS

Title: Fast Analog Computing with Emergent Transient States

Milestone N^o: M8-4

Due date: 31/05/2009

Delivery Date: 09/06/2009

Short Description:

The third public release of PyNN (version 0.5) was on 09/06/2009. The software is available to download from http://neuralensemble.org/PyNN.

Development of FacetsML (http://neuralensemble.org/trac/FacetsML) has been discontinued, since PyNN now has a module for exporting NeuroML (http://www.neuroml.org) directly, and the neuroConstruct software (http://www.neuroconstruct.org) can now export NeuroML software descriptions in PyNN format. These direct pathways between PyNN and NeuroML make the indirect pathway via FacetsML unnecessary.

This report summarizes the changes to PyNN since the previous release.

Partners owning: CNRS-UNIC

Partners contributed: CNRS-UNIC, CNRS-INCM, ALUF, EPFL, UHEI, TUG, INRIA

Made available to: Public

1 Introduction

PyNN is a Python package for simulator-independent specification of neuronal network models. It is being widely used within the FACETS consortium and forms a major part of the bridge between software simulations and the FACETS neuromorphic hardware systems. It is also starting to attract users from outside FACETS.

FacetsML is a declarative equivalent to PyNN, using XML, but its use as a bridge to NeuroML has been superseded by the development of direct links between PyNN and NeuroML and it is no longer being developed.

The first public release of PyNN (version 0.3) was on 24/05/2007, corresponding to FACETS Milestone M8-2. Deliverable D8-1 gives more information about PyNN, including the motivation for the development of PyNN and FacetsML. The second public release of PyNN (version 0.4) was on 30/05/2008. The changes in this version were listed in Deliverable D8-3, which also described FacetsML. This report summarizes the changes to PyNN in version 0.5.

2 Changes in version 0.5

There have been rather few changes to the API in this version, which has focused rather on improving the simulator interfaces and on an internal code-reorganization which aims to make PyNN easier to test, maintain and extend.

2.1 Principal API changes

- Removed the 'string' connection methods from the Projection constructor.
- The method argument now must be a Connector object, not a string.
- Can now record synaptic conductances.
- Can now access weights and delays of individual connections one-at-a-time within a Projection through Connection objects.
- Added an interface for injecting arbitrary time-varying currents into cells.
- Added `get_v()` and `get_gsyn()` methods to the Population class, enabling membrane potential and synaptic conductances to be read directly into memory,rather than saved to file.

2.2 Improvements to simulator back-ends

- Implemented an interface for the Brian simulator.
- Re-implementated the interface to NEURON, to use the new functionality in v7.0.
- Removed support for version 1 of NEST. The module for NEST v2 is now simply called pyNN.nest.
- The PCSIM implementation is now more complete, and more compatible with the other back-ends.
- Behind-the-scenes refactoring to implement the API in terms of a small number of low-level, simulator-specific operations. This reduces redundancy between simulator modules, and makes it easier to extend PyNN, since if new functionality uses the low-level operations, it only needs to be written once, not once for each simulator.

2.3 Detailed list of changes

```
* Renamed ‘nest2‘ to ‘nest‘.
* Random number generators  now "parallel_safe" by default.
* Added documentation on running parallel simulations.
* Trying a new method of getting the last data points out of NEST: we always simulate to ‘t+dt‘. This should be fine
  until we implement the possibility of accessing Vm directly from the ‘ID‘ object (see ticket:35). In the meantime,
  hopefully the NEST guys will change this behaviour.
* ‘gather=True‘ now works for all modules, even without a shared filesystem (requires ‘mpi4py‘).
* Added an ‘allow_update_on_post‘ option to the NEURON weight adjuster mechanisms. This is set to 0 (false) for
  consistency with NEST, which means that weight updates are accumulated and are applied only on a pre-synaptic
  spike, although I’m not yet sure (a) what the correct behaviour really is, (b) what PCSIM and Brian do.
* The ‘pcsim‘ module is now implemented in terms of the common implementation, using just ‘net.add()‘ and
  ‘net.connect()‘. I have just commented out the old code (using ‘CuboidGridObjectPopulation‘s and
  ‘ConnectionsProjection‘s) rather than deleting it, as it will probably be mostly re-used when optimizing later.
* ‘Population.__getitem__()‘ now accepts slices (see ticket:21).
* NEST does not record values at t=0 or t=simtime so, for compatibility with the other simulators, we now add these
  values manually to the array/datafile.
```

* Fixed the ‘SpikeSourcePoisson‘ model in the ‘neuron‘ module so it has a really fixed duration, not just an ’on average’ fixed duration.
* Created a new Exception type, ‘RecordingError‘, for when we try to record membrane potential from a ‘SpikeSource‘.
* Renamed the ‘param_dict‘ argument of ‘create()‘ to ‘cellparams‘, for consistency with ‘Population.__init__()‘.
* Created default implementations of nearly all functions and classes in ‘common‘, some of which depend on the simulator package having a ‘simulator‘ module that defines certain ’primitive’ capabilities.
* Created default implementations of all ‘Connector‘ classes in ‘connectors‘, which depend on the ‘Projection‘ having a ‘ConnectionManager‘ which in turn has a ‘connect()‘ method implementing divergent connect.
* Added a ‘ConnectionManager‘ class to the ‘simulator‘ module in ‘nest2‘, ‘neuron‘ and ‘brian‘ packages. This allows (i) a common way of managing connections for both the ‘connect()‘ function and the ‘Projection‘ class, (ii) a common specification of ‘Connector‘ algorithms in terms of method calls on a ‘ConnectionManager‘ instance.
* Added ‘weights_iterator()‘ and ‘delays_iterator()‘ to the base ‘Connector‘ class, to make them available to all simulator modules.
* Moved ‘Connector‘ base classes from ‘common‘ into a new module, ‘connectors‘.
* Moved standard dynamic synapse base classes from ‘common‘ into a new module, ‘synapses‘.
* Moved standard cell base classes from ‘common‘ into a new module, ‘cells‘.
* Moved ‘Timer‘ class from ‘common‘ to ‘utility‘.
* ‘Population‘ attributes ‘all_cells‘ and ‘local_cells‘ are now an official part of the API (‘cell‘ is as an alias for ‘local_cells‘ for now since it was widely used, if not officially part of the API, in 0.4).
* Removed the ’string’ connection methods from the ‘Projection‘ constructor. The ‘method‘ argument now *must* be a ‘Connector‘ object, not a string.
* Standard cell types now know what things can be recorded from them (‘recordable‘ attribute).
* Added new Exception ‘NothingToWriteError‘. Calling ‘printSpikes()‘, etc, when you have not recorded anything raises an Exception, since this is quite likely a mistake, but it needs to be a specific Exception type so it can be handled without inadvertently catching all other errors that are likely to arise during writing to file.
* Moved old tests to examples folder
* Added Padraig Gleeson’s modifications to neuroml.py
* little change in setup of the rng_seeds, re-add the possibility to give a seed for a RNG that then draws seeds for the simulation. In that way one does not have to provide the exact number of seeds needed, just one.
* add ‘save_population()‘ and ‘load_population()‘ functions to ‘utility‘.
* ‘test/explore_space.py‘ is now working. The test script saves results to a NeuroTools datastore, which the ‘explore_space‘ script can later retrieve. Only plotting does not work in distributed mode due to lack of X-display.
* STDP testing improved. ‘nest‘, ‘pcsim‘ and ‘neuron‘ now give pretty similar results for ‘SpikePairRule‘ with ‘AdditiveWeightDependence‘. I think some of the remaining differences are due to sampling the weights every millisecond, rather than at the beginning of each PSP. We need to develop an API for recording weights in PyNN (PCSIM and NEURON can record the weights directly, rather than by sampling, not sure about NEST).
  However, I suspect there are some fundamental differences in the algorithms (notably *when* weight changes get applied), that may make it impossible to completely reconcile the results.
* Moved the ‘MultiSim‘ class from ‘test/multisim.py‘ into the ‘utility‘ module.
* The ‘pcsim‘ module now supports dynamic synapses with both conductance-based and current-based synapses.
* ‘Projection.saveConnections()‘, ‘printWeights()‘ and ‘weightHistogram()‘ now work in the ‘pcsim‘ module.
* ‘pcsim‘ ‘Connector‘s now handle arrays of weights/delays.
* Implemented ‘FromListConnector‘ and ‘FromFileConnector‘ for ‘pcsim‘.
* Changed tau_ref for hardware neuron model from 0.4ms to 1.0ms. Old estimation was distorted by hardware error.
* Fixed a bug whereby spikes from a ‘Population‘ of ‘SpikeSourceArray‘s are not recorded if they are set after creation of the population.
* Optimisations to improve the building times of large networks in ‘nest2‘.
* Can now record synaptic conductances.
* Added an interface for the Brian simulator.
* When you try to write data to a file, any existing file of the same name is first renamed by appending ’_old’ to the filename.
* Modification of the RandomDistribution object to allow the specification of boundaries, and the way we deal with numbers drawn outside those boundaries. Numbers may be clipped to min/max values, or either redrawn till they fall within min/max values
* Added the possibility to select a particular model of plasticity when several are available for the same plastic rule. This is mainly used in NEST, where we set the stdp_synapse_hom as the default type, because it is more efficient when (as is often the case) all plasticity parameters are identitical.
* Added the possibility of giving an expression for distant-dependent weights and delays in the ‘DistanceDependentProbabilityConnector‘.
* Harmonization of ‘describe()‘ methods across simulators, by moving its definition into ‘common‘. ‘describe()‘ now returns a string, rather than printing directly to stdout. This lets it be used for writing to log files, etc. You will now have to use ‘print p.describe()‘ to obtain the old behaviour.‘describe()‘ now also takes a ‘template‘ argument, allowing the output to be customized.
* Added an interface for injecting arbitrary currents into cells.
  Usage example:

```
cell = create(IF_curr_exp)
current_source1 = DCSource(amplitude=0.3, start=20, stop=80)
current_source2 = StepCurrentSource(times=[20,40,60,80], amplitudes=[0.3,-0.3,0.3,0.0])
cell.inject(current_source1)            # two alternatives
current_source2.inject_into([cell])
```

  ‘DCSource‘ and ‘StepCurrentSource‘ are available in the ‘nest2‘, ‘neuron‘ and ‘brian‘ modules.

‘NoisyCurrentSource‘ is only available in ‘nest2‘ for the time being, but it will be straightforward to add it for the other backends. Adding ‘ACSource‘, etc., should be straightforward.
* Optimised setting of parameter values by checking whether the list of parameters to be set contains any computed parameters, and only getting/translating all parameters in this case. Before, each time we wanted to set a parameter, we always got all the native_parameters and translated them even when there was a one-to-one correspondence between parameters.
* Implemented the Gutig rule and the van Rossum rule of plasticity in ‘nest2‘, and changed the old naming.
* In ‘nest2‘, fixed also the meanSpikeCount() method to directly obtain the spike count from the recorder, and not from the file.
* Great improvement of the distance dependant distance. Speed up considerably the building time.
* Moved unit tests into their own subdirectory
* Fixed a bug in the periodic boundary conditions in ‘DistanceDependentProbabilityConnector‘ class if they are specified by the user and not linked to the grid sizes.
* Reduced the number of adjustable parameters in the ‘IF_facets_hardware1‘ standard cell model.
* Reimplemented the ‘neuron‘ module to use the new features of NEURON (‘HocObject‘, etc) available in v7.0.
* Added ‘get_v()‘ method to the ‘Population‘ class, enabling membrane potential to be read directly into memory, rather than saved to file.