Current status and future plans for NeuroTools

Pierre Yger

BioEngineering Department, Imperial College, London

March 15, 2012
The curse of the data

• Simulations and/or multiple recordings are nowadays common.
• Hundreds, thousands, even hundreds of thousands recordings.
• More and more complex analysis handling those massive data.

[Smith et al, 2008] [Blanche et al, 2005] [Izhikevich et al, 2007]
Analysis workflows

Direct consequence of this complexity:

- Analysis/Workflows has to be standardised
- It’s harder to be sure your code is doing what you want it to do
Analysis workflows

Direct consequence of this complexity:

→ Analysis/Workflows has to be standardised
→ It’s harder to be sure your code is doing what you want it to do

Several solutions to face this increase in complexity:
Analysis workflows

Direct consequence of this complexity:

→ Analysis/Workflows has to be standardised
→ It’s harder to be sure your code is doing what you want it to do

Several solutions to face this increase in complexity:

→ Work more: harder, better, stronger
Analysis workflows

Direct consequence of this complexity:

→ Analysis/Workflows has to be standardised
→ It’s harder to be sure your code is doing what you want it to do

Several solutions to face this increase in complexity:

→ Work more: harder, better, stronger
→ Work more with people you trust (shared project)
Analysis workflows

Direct consequence of this complexity:

→ Analysis/Workflows has to be standardised
→ It’s harder to be sure your code is doing what you want it to do

Several solutions to face this increase in complexity:

→ Work more: harder, better, stronger
→ Work more with people you trust (shared project)
→ Work less by using already coded tools (Let’s trust again)
Analysis workflows

Direct consequence of this complexity:

→ Analysis/Workflows has to be standardised
→ It’s harder to be sure your code is doing what you want it to do

Several solutions to face this increase in complexity:

→ Work more: harder, better, stronger
→ Work more with people you trust (shared project)
→ Work less by using already coded tools (Let’s trust again)
→ Test or share your code with others to increase the confidence in it.
Analysis workflows

Direct consequence of this complexity:
→ Analysis/Workflows has to be standardised
→ It’s harder to be sure your code is doing what you want it to do

Several solutions to face this increase in complexity:
→ Work more: harder, better, stronger
→ Work more with people you trust (shared project)
→ Work less by using already coded tools (Let’s trust again)
→ Test or share your code with others to increase the confidence in it.

Solutions:
Simplify reuse of code by new tools/methods (svn, documentation, tests, well-defined API) and common format
NeuroTools was initiated during the FACETS projects, aiming to:

1. increase the productivity of modellers by automating, simplifying, and establishing best-practices for common tasks
2. increase the productivity of the modelling community by reducing code duplication
3. increase the reliability of the tools, leveraging Linus’s law: “given enough eyeballs, all bugs are shallow”

**Current Status:**
Still not 'stable', not modular enough, should be simplified.
The need for a common format

Some Simulators
- Brian brian.di.ens.fr/
- Catacomb www.catcmb.org
- CSIM www.lsm.tugraz.at/csim
- GENESIS www.genesis-sim.org
- Matlab www.mathworks.com
- Mvaspike mvaspike.gforge.inria.fr
- Neosim www.neurogems.org/neosim2
- NEST www.nest-initiative.org
- NEURON www.neuron.yale.edu
- Neurospaces neurospaces.sourceforge.net
- SpikeNET www.spikenet-technology.com
- SPLIT
- Topographica topographica.org
- Your home made one
- ...

Some Analysis tools
- Spike Train Analysis Toolkit neuroanalysis.org/toolkit/intro.html
- Spike Toolbox www.ini.uzh.ch/~dylan/spike_toolbox
- MEA-Tools material.brainworks.uni-freiburg.de
- Spike train analysis software www.blki.hu/~szucs/OS3.html
- NeuroExplorer www.adinstruments.com
- Spike Train Analysis with R (STAR) sites.google.com/site/spiketrainanalysiswithr/
- OpenElectrophy http://neuralensemble.org/trac/OpenElectrophy
- FIND http://find.bccn.uni-freiburg.de/
- Your home made one
- ...

try: reduce(duplication, community)
neo: the chosen one

- generic container
- extensible
- r/w common formats
- handle quantities
- match various needs
  - real recordings
  - simulations
- link with OpenElectrophy
- (wait for tomorrow)
The NeuroTools Structure

- Particular attention on documentation, to make functions usable
- Tests tend to be systematic (currently > 80% of coverage)
NeuroTools.stgen

Efficient generation of time varying signals
• (in)homogeneous poisson/gamma processes
• Orstein Ulbeck processes
• Shot noise
• ...
NeuroTools.parameters

Deal with the parameter mess in simulations

• Good practice to separate the parameters from the model itself.
• At least, parameters should be in a separate section of a file.

Advantages

→ Helps version control, as model vs parameter changes can be conceptually separated
→ Make it easier to track a simulation project, since the parameter sets can be stored in a database, displayed in a GUI, etc.
→ Consolidate the reproducibility of the results (alternatives: sumatra)
The ParameterSet class

ParameterSet objects may be created from a dict:

```python
>> sim_params = ParameterSet({'dt': 0.11, 'tstop': 1000.0})
```

They may be nested:

```python
>> I_params = ParameterSet({'tau_m': 15.0, 'cm': 0.75})
>> network_params = ParameterSet({
...   'excitatory_cells': E_params,
...   'inhibitory_cells': I_params})
>> P = ParameterSet({'sim': sim_params,
...                   'network': network_params,
...                   label="my_params")
```
Parameter spaces

```
>>> P = ParameterSpace(
...     {'cm': 1.0,
...      'tau_m': ParameterRange([10.0, 15.0, 20.0])
...     })
>>> for p in P.iter_inner():
...     print p
...     print p
...
{'tau_m': 10.0, 'cm': 1.0}
{'tau_m': 15.0, 'cm': 1.0}
{'tau_m': 20.0, 'cm': 1.0}
```
>> P = ParameterSpace({
...       'cm': 1.0,
...       'tau_m': NormalDist(mean=12.0, std=5.0)
...     })
>> for p in P.realize_dists(2):
...     print p
...
{‘tau_m’: 20.237970275471028, ‘cm’: 1.0}
{‘tau_m’: 10.068110582245506, ‘cm’: 1.0}
NeuroTools.signals

Dealing with event signals:
- SpikeTrain
- SpikeList

And with analog signals
- AnalogSignal
- AnalogSignalList
  - MembraneTraceList
  - CurrentTraceList
  - ConductanceTraceList

→ All merged into a single class Segment to match the neo syntax
The SpikeTrain objects

Object to handle the spikes produced by one cell during \([t_{\text{start}}, t_{\text{stop}}]\)

- \text{duration()}, \text{time\_slice()}, \text{time\_offset()}
- \text{isi()}, \text{mean\_rate()}, \text{cv\_isi()}
- \text{raster\_plot()}
- \text{time\_histogram()}, \text{psth()}
- \text{distance\_victorpurpura()}, \text{distance\_kreuz()}
- \text{merge()}
- ...

→ Distances should be separated
→ Functions instead of methods for less code duplication
The SpikeList class

object to handle the spikes produced by several cells during \([t_{\text{start}}, t_{\text{stop}}]\)

- More or less a dictionary of SpikeTrains
- Cells have unique id
- They could be arranged on a grid for graphical purpose

```python
>>> spikes = SpikeList(data, id_list=range(10000), t_start=0,
                      t_stop=500, dims=[100,100])
>>> spikes[245].mean_rate()
```
The SpikeList class

- All SpikeTrain functions can be called
- Easy way of slicing, either by id, time or even by user-defined conditions.
- Easy way of building SpikeTrain from your own fileformats
- Pairs generators to average functions over custom-defined pairs:
  - `pairwise_cc()`, `pairwise_pearson_corrcoeff()`, ...
- Graphical functions: `raster_plot()`, activity maps and movies for 2D SpikeList, ...
The SpikeList class

```python
>> all_spikes = load_spikelist('data.gdf', t_start=0, t_stop=500, dims=[65,65])
>> ids = all_spikes.select_ids('cell.mean_rate() > 10')
>> my_spikes = all_spikes.id_slice(ids)
>> my_spikes.firing_rate(time_bin=5, display=subplot(131))
>> my_spikes.raster_plot(1000, display=subplot(132))
>> my_spikes.activity_map(display=subplot(133))
```
The SpikeList class

Pairs Selectors: Random, Auto, DistantDependent, ...

```python
>> pairs = RandomPairs(all_spikes, all_spikes, no_silent=True)
>> spikes.pairwise_cc(5000, pairs, time_bin=5)
>> x = spikes.pairwise_pearson_corrcoeff(5000, pairs, time_bin=5)
>> hist(x, 100)
```
The AnalogSignal(List) class

Object to handle analog signals produced during \([t_{\text{start}}, t_{\text{stop}}]\), with sampling time \(dt\).

- duration(), time_slice(), time_offset()
- threshold_detection, event_triggered_average()
- slice_by_events()
- ...

```python
>> signal = sin(arange(0, 1000, 0.1))
>> x = AnalogSignal(signal, dt=0.1)
>> spk = SpikeTrain(arange(0,1000,100))
>> x.event_triggered_average(spk, average=False, t_min=20, t_max=20)
```
New syntax

After the CodeJam, we should have:

```python
>> data = neo.PyNNNumpyI0("simulation.npy")
>> data.read_segment()
>> mean_rate(data)
>> psth(data, events=[10, 250, 350])
>> subdata = data.time_slice(2000, 5000)
>> raster_plot(subdata)
```
Further extensions

• Consolidate the NeuroTools structure:
  ○ Simplify the API
  ○ Finish the transition towards Segment objects
  ○ Clarify all dependencies and simplify the addition of extra functions.
  ○ Have a look around (nibabel/nipy fMRI community)

• Add more sophisticated analysis functions:
  ○ More sharing and reuse of code
  ○ Gain in confidence and correctness

• Enlarge the community

We Want YOU!
Questions?

To give a try: http://www.neuralensemble.org
Download, install, play, and contribute!

Contributors
Daniel Brüederle, Andrew Davison, Samuel Garcia, Jens Kremkow, Eilif Muller, Laurent Perrinet, Michael Schmucker