Challenges and solutions in replicability and provenance tracking for simulation projects

Andrew P. Davison

Unité de Neurosciences, Information et Complexité
Centre National de la Recherche Scientifique
Gif sur Yvette, France

andrew.davison@unic.cnrs-gif.fr
http://www.andrewdavison.info
@apdavison

Why is reproducing a simulation exactly so hard?

- complexity of code
- complexity of computing environment
- rapid change (in own code and in dependencies)
- difficulty in capturing every piece of information necessary to reproduce a simulation

What can we do about it?

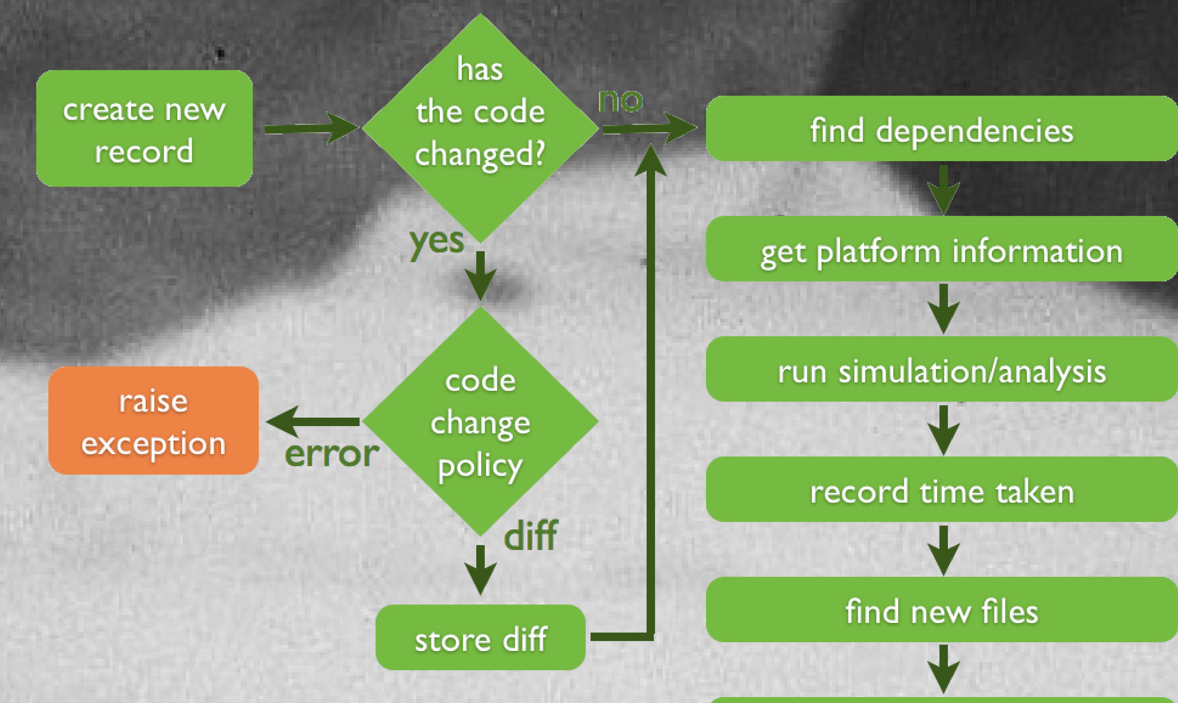
- reduce the complexity of our code through good software engineering practices
- use version control (Subversion, Mercurial, Git, etc.)
- develop tools to automatically and systematically capture every detail of our simulations and their environment: an automated lab notebook

What do we need to capture?

- the code that was run
  - the executable (version, library versions, compiler, compilation options)
  - the script (repository URL, version, imported modules/packages and their versions)
- how it was run: parameter files, command-line options
- the platform on which it was run: processor architecture, operating system, cluster architecture
- why was it run?
- what was the outcome?

Requirements

- support many different workflows: command-line, GUI, batch jobs, solo or collaborative
- automate as much as possible, prompt the user for the rest
- interact with version control systems
- link to data generated by the simulation
- support all and any command-line driven simulators
- support both local and networked storage of data
- be easily extensible and configurable to different workflows
- **be very easy to use, or only the very conscientious will use it**



create new record
has the code changed?
no
yes
code change policy
error
raise exception
diff
store diff
find dependencies
get platform information
run simulation/analysis
record time taken
find new files
add tags
save record

Control flow when running a simulation with Sumatra

Running simulations using Sumatra

```
$ cd myproject
$ smt init MyProject
$ nrniv main.hoc default.param
$ smt run --executable=nrniv --main=main.hoc default.param
$ smt configure --simulator=nrniv --main=main.hoc
$ smt run default.param
$ smt run -n 10 default.param
$ smt list
20100715-174949
20100715-175111
$ smt run --label=haggling --reason="determine whether the
gourd is worth 3 or 4 shekels" romans.param
$ smt comment "apparently, it is worth NaN shekels."
$ smt tag "Figure 6"
$ smt run --reason="test effect of a smaller time constant"
default.param tau_m=10.0
$ smt repeat haggling
The simulation results match.
$ smtweb 8000 &
```

how you run your simulation normally

how to run it with Sumatra

set defaults so you don't have to do so much typing

that's better

run a distributed simulation with MPI

use smt list -l for more details

Sumatra generates labels automatically based on the timestamp

or you can specify your own label

tell Sumatra why you did the simulation (invaluable 6 months later when you're writing the paper)

and summarize the outcome

add tags to help you find it later

change parameters on the command line

just to check. If they don't match, Sumatra will show you exactly what changed

start a local webserver for the web interface

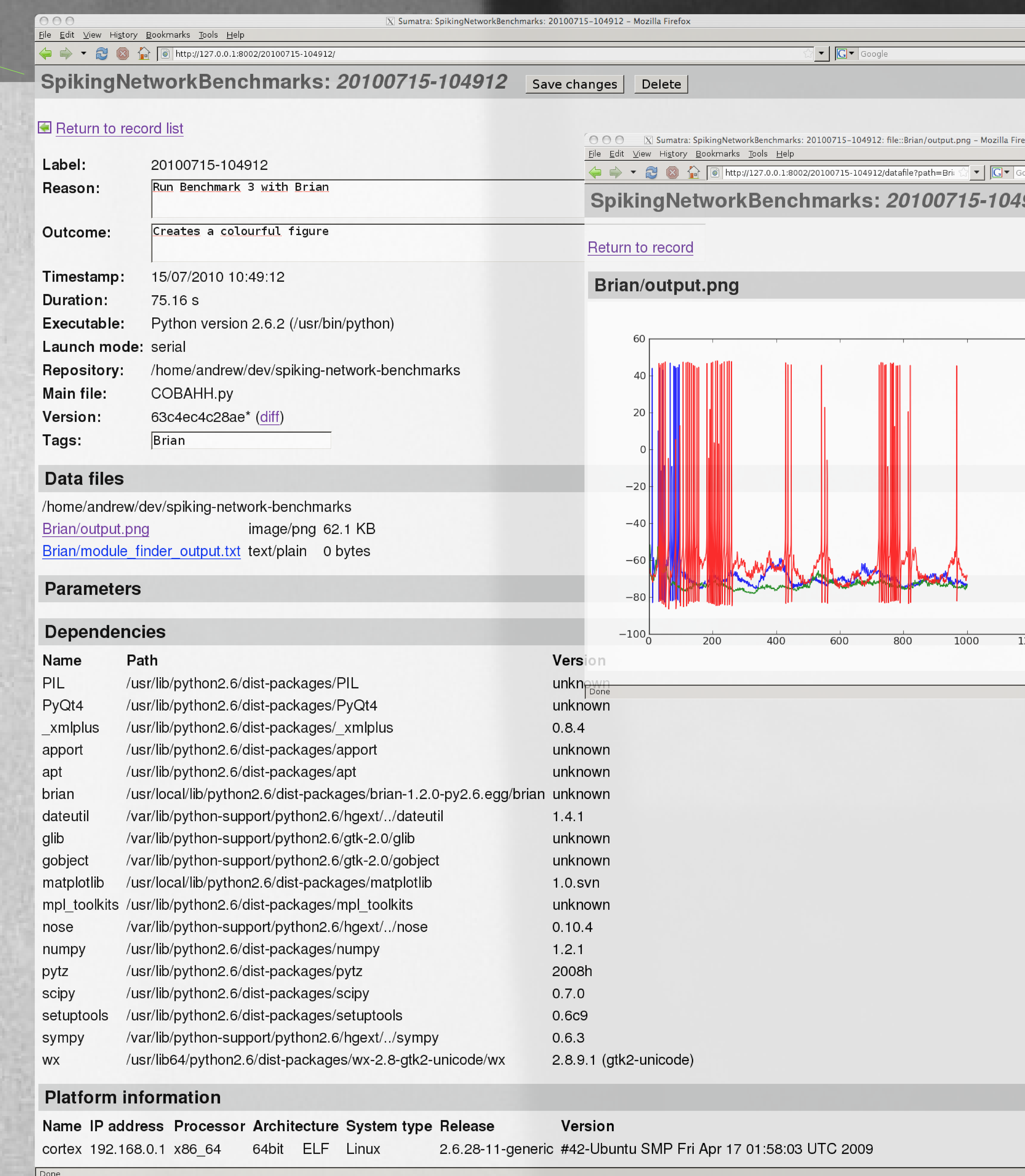
"I thought I used the same parameters but I'm getting different results"

"I can't remember which version of the code I used to generate figure 6"

"The new student wants to reuse that model I published three years ago but he can't reproduce the figures"

"It worked yesterday"

"Why did I do that?"



Sumatra: SpikingNetworkBenchmarks: 20100715-104912 – Mozilla Firefox

http://127.0.0.1:8002/20100715-104912/

SpikingNetworkBenchmarks: *20100715-104912* Save changes Delete

Return to record list

Label: 20100715-104912
Reason: Run Benchmark 3 with Brian
Outcome: Creates a colourful figure
Timestamp: 15/07/2010 10:49:12
Duration: 75.16 s
Executable: Python version 2.6.2 (/usr/bin/python)
Launch mode: serial
Repository: /home/andrew/dev/spiking-network-benchmarks
Main file: COBAHH.py
Version: 63c4ec4c28ae* (diff)
Tags: Brian

Data files

/home/andrew/dev/spiking-network-benchmarks

Brian/output.png	image/png	62.1 KB
Brian/module_finder_output.txt	text/plain	0 bytes

Parameters

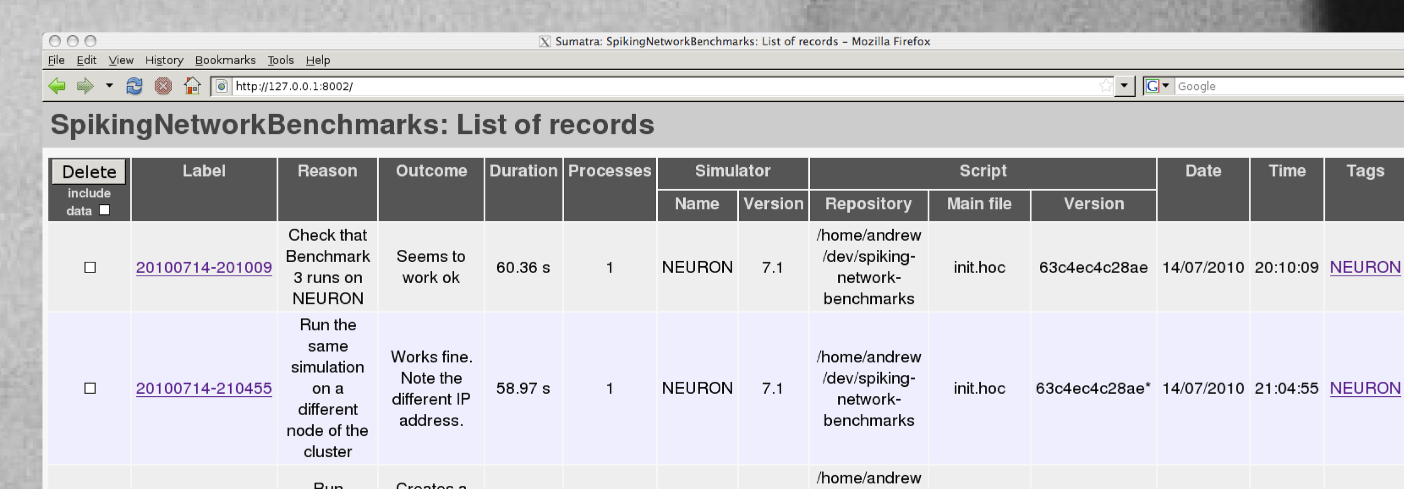
Dependencies

Name	Path	Version
PIL	/usr/lib/python2.6/dist-packages/PIL	unknown
PyQt4	/usr/lib/python2.6/dist-packages/PyQt4	unknown
_xmlplus	/usr/lib/python2.6/dist-packages/_xmlplus	0.8.4
apport	/usr/lib/python2.6/dist-packages/apport	unknown
apt	/usr/lib/python2.6/dist-packages/apt	unknown
brian	/usr/local/lib/python2.6/dist-packages/brian-1.2.0-py2.6.egg/brian	unknown
dateutil	/var/lib/python-support/python2.6/hgext/../dateutil	1.4.1
glib	/var/lib/python-support/python2.6/gtk-2.0/glib	unknown
gobject	/var/lib/python-support/python2.6/gtk-2.0/gobject	unknown
matplotlib	/usr/local/lib/python2.6/dist-packages/matplotlib	1.0.svn
mpl_toolkits	/usr/lib/python2.6/dist-packages/mpl_toolkits	unknown
nose	/var/lib/python-support/python2.6/hgext/../nose	0.10.4
numpy	/usr/lib/python2.6/dist-packages/numpy	1.2.1
pytz	/usr/lib/python2.6/dist-packages/pytz	2008h
scipy	/usr/lib/python2.6/dist-packages/scipy	0.7.0
setuptools	/usr/lib/python2.6/dist-packages/setuptools	0.6c9
sympy	/var/lib/python-support/python2.6/hgext/../sympy	0.6.3
wx	/usr/lib64/python2.6/dist-packages/wx-2.8-gtk2-unicode/wx	2.8.9.1 (gtk2-unicode)

Platform information

Name	IP address	Processor	Architecture	System type	Release	Version
cortex	192.168.0.1	x86_64	64bit ELF	Linux	2.6.28-11-generic	#42-Ubuntu SMP Fri Apr 17 01:58:03 UTC 2009

Sumatra: SpikingNetworkBenchmarks: 20100715-104912: file::Brian/output.png – Mozilla Firefox
SpikingNetworkBenchmarks: 20100715-104912
Return to record
Brian/output.png
60
40
20
0
−20
−40
−60
−80
−100
200
400
600
800
1000
1200

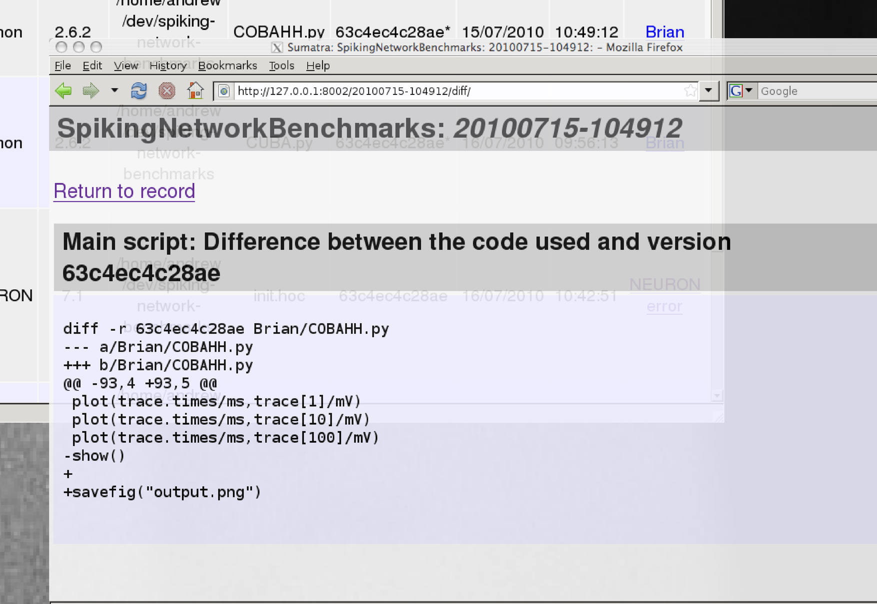


Sumatra: SpikingNetworkBenchmarks: List of records – Mozilla Firefox

http://127.0.0.1:8002/

SpikingNetworkBenchmarks: List of records

Delete (include data)	Label	Reason	Outcome	Duration	Processes	Simulator Name	Simulator Version	Script Repository	Script Main file	Script Version	Date	Time	Tags
☐	20100714-201009	Check that Benchmark 3 runs on NEURON	Seems to work ok	60.36 s	1	NEURON	7.1	/home/andrew /dev/spiking-network-benchmarks	init.hoc	63c4ec4c28ae	14/07/2010	20:10:09	NEURON
☐	20100714-210455	Run the same simulation on a different node of the cluster	Works fine. Note the different IP address.	58.97 s	1	NEURON	7.1	/home/andrew /dev/spiking-network-benchmarks	init.hoc	63c4ec4c28ae*	14/07/2010	21:04:55	NEURON
☐	20100715-104912	Run Benchmark 3 with Brian	Creates a colourful figure	75.16 s	1	Python	2.6.2	/home/andrew /dev/spiking-	COBAHH.py	63c4ec4c28ae*	15/07/2010	10:49:12	Brian
☐	20100716-095613	Run Benchmark 2 with Brian	Graph shows smoothed firing rate as a function of time	5.68 s	1	Python	[illegible]						
☐	20100716-104251	Run Benchmark 3 on four nodes	FAILED: complains that delay is zero. Needs to be non-zero for distributed simulations.	0.89 s	4	NEURON							
		Run											



Sumatra: SpikingNetworkBenchmarks: 20100715-104912: – Mozilla Firefox

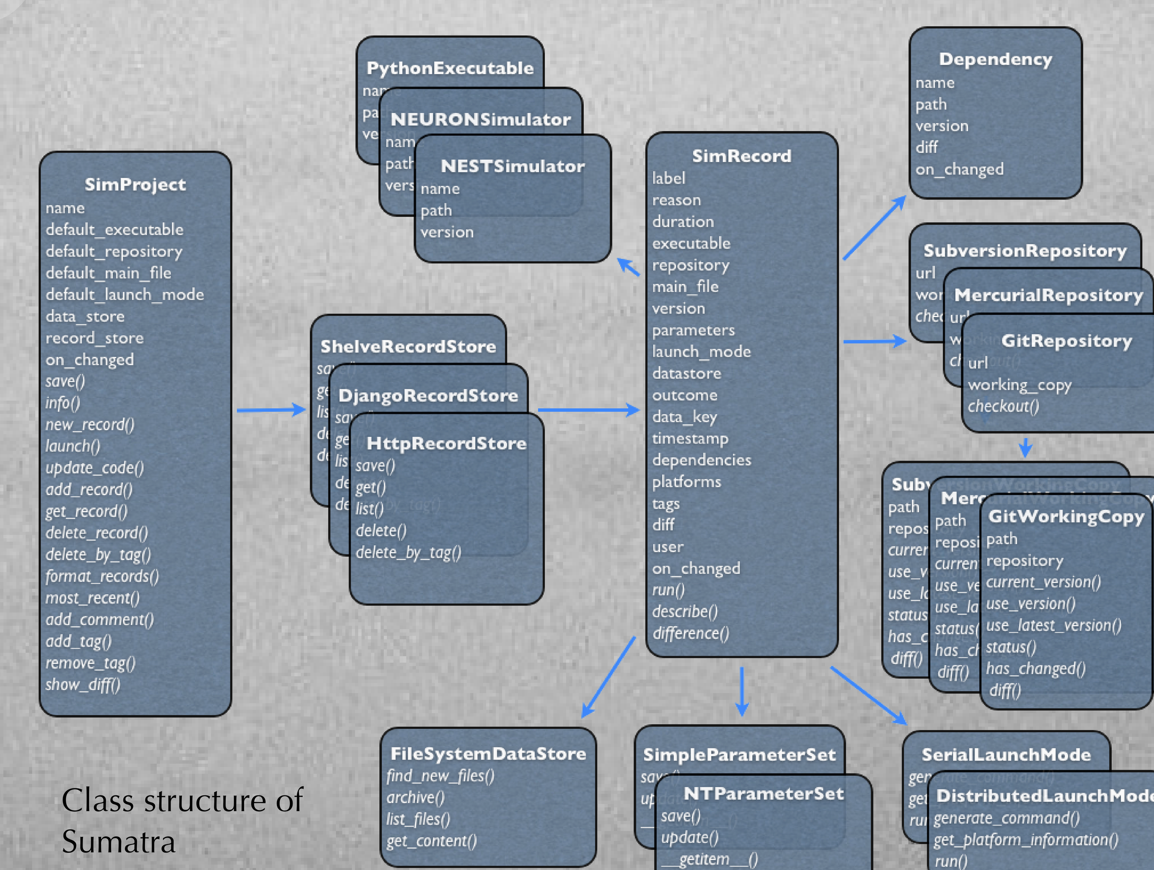
http://127.0.0.1:8002/20100715-104912/diff/

SpikingNetworkBenchmarks: *20100715-104912*

Return to record

Main script: Difference between the code used and version 63c4ec4c28ae

```
diff -r 63c4ec4c28ae Brian/COBAHH.py
--- a/Brian/COBAHH.py
+++ b/Brian/COBAHH.py
@@ -93,4 +93,5 @@
 plot(trace.times/ms,trace[1]/mV)
 plot(trace.times/ms,trace[10]/mV)
 plot(trace.times/ms,trace[100]/mV)
-show()
+
+savefig("output.png")
```



SimProject
name
default_executable
default_repository
default_main_file
default_launch_mode
data_store
record_store
on_changed
save()
info()
new_record()
launch()
update_code()
add_record()
get_record()
delete_record()
delete_by_tag()
format_records()
most_recent()
add_comment()
add_tag()
remove_tag()
show_diff()
PythonExecutable
NEURONSimulator
NESTSimulator
name
path
version
ShelveRecordStore
DjangoRecordStore
HttpRecordStore
save()
get()
list()
delete()
delete_by_tag()
SimRecord
label
reason
duration
executable
repository
main_file
version
parameters
launch_mode
datastore
outcome
data_key
timestamp
dependencies
platforms
tags
diff
user
on_changed
run()
describe()
difference()
Dependency
name
path
version
diff
on_changed
SubversionRepository
MercurialRepository
GitRepository
url
working_copy
checkout()
GitWorkingCopy
path
repository
current_version()
use_version()
use_latest_version()
status()
has_changed()
diff()
FileSystemDataStore
find_new_files()
archive()
list_files()
get_content()
SimpleParameterSet
NTParameterSet
save()
update()
__getitem__()
SerialLaunchMode
DistributedLaunchMode
generate_command()
get_platform_information()
run()

Class structure of Sumatra

Background

Reproducibility of experiments is one of the foundation stones of science. A related concept is provenance, being able to track a given scientific result, such as a figure in an article, back through all the analysis steps (verifying the correctness of each) to the original raw data, and the experimental protocol used to obtain it. In computational, simulation-based science, reproduction of previous experiments, and establishing the provenance of results, ought to be easy, given that computers are deterministic, not suffering from the problems of inter-subject and trial-to-trial variability that make reproduction of biological experiments more challenging. In general, however, it is not at all easy.

It is useful to distinguish between independent reproduction of an experiment by an independent researcher, and replication of an experiment, using the same code, perhaps some months or years later. Independent reproducibility is the gold standard, and difficulties in achieving it arise mainly from the challenges in describing complex models and simulation experiments in sufficient detail (Nordlie et al., 2009). Replicability is also important, however, not least in determining whether the failure of others to reproduce a result is due to errors in the original code, and is not necessarily any more frequently achieved. The difficulties of replicability are an underappreciated problem in computational neuroscience, one that will only become more important as the ambition of computational neuroscience and the scrutiny placed upon science in general (*cf* the recent media attention to problems of replicability in climate science simulations) grow.

The failure to routinely achieve replicability in computational science in general (Donoho et al., 2009) is probably due to the complexity and rapidity of change of our code and our computing environments, and the difficulty of capturing every essential piece of information needed to reproduce a computational experiment using existing tools (spreadsheets, version control systems and paper notebooks). In other areas of science, particularly in applied science laboratories with high-throughput, highly-standardised procedures, electronic lab notebooks and laboratory information management systems (LIMS) are in widespread use, but none of these tools seem to be well suited for tracking simulation experiments. VisTrails (http://www.vistrails.org/) is a promising tool for visually creating and running computational workflows with provenance tracking, but its use requires totally recreating your current workflow within the VisTrails environment.

In developing a tool for tracking simulation experiments, something like an electronic lab notebook for computational science, there are a number of challenges: (i) different researchers have very different ways of working and different workflows: command line, GUI, batch-jobs (e.g. in supercomputer environments), or any combination of these for different components (simulation, analysis, graphing, etc.) and phases of a project; (ii) some projects are essentially solo endeavours, others collaborative projects, possibly distributed geographically; (iii) as much as possible should be recorded automatically; if it is left to the researcher to record critical details there is a risk that some details will be missed or left out, particularly under pressure of deadlines.

Here I present a proposed solution to the problems of ensuring reproducibility and tracking the provenance of results, that addresses the challenges outlined above. This solution consists of a core library, implemented as a Python package, Sumatra, together with a series of interfaces that build on top of this: a command-line interface and web interface are provided; it could also be incorporated in existing graphical environments such as neuroConstruct or nrngui. Each of these interfaces enables (i) launching simulations with automated recording of provenance information; and (ii) managing a simulation project: browsing, viewing, deleting simulations. Alternatively, modellers using Python can use the Sumatra package directly in their own code then simply launch simulations in their usual way. Sumatra is distributed as open-source software (http://neuralensemble.org/sumatra/) and is developed using a community model: anyone is welcome to get involved with its development.

References

Nordlie E., Gewaltig M-O, Plesser HE (2009): Towards Reproducible Descriptions of Neuronal Network Models. PLoS Computational Biology. 5:e1000456. http://dx.doi.org/10.1371/journal.pcbi.1000456

Donoho DL, Maleki A, Rahman IU, Shahram M, and Stodden V (2009). 15 years of reproducible research in computational harmonic analysis. Computing in Science & Engineering, 11:8–18. http://dx.doi.org/10.1109/MCSE.2009.15

Sumatra is something like an automated lab notebook for simulation-based science, designed to have minimal impact on your current workflow.

Documentation and downloads from:

http://neuralensemble.org/sumatra/

Background image: "lab notebook" by benjaminlansky http://www.flickr.com/photos/77443319@N08/3110638201/