Sumatra
an electronic lab notebook for simulation projects

Andrew Davison
UNIC, CNRS

FACETS CodeJam #3
Freiburg, 7th-9th October 2009

http://neuralensemble.org/trac/sumatra
This presentation is licenced under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 licence
http://creativecommons.org/licenses/by-nc-sa/3.0/
Reproducibility
“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?”
Why isn’t it easy to reproduce a computational experiment exactly?
What can we do about it?
An automated lab notebook to record every detail of our simulations
What do we need to record?

- the code that was run
- how it was run (parameter files, command-line options)
- the platform on which it was run
- why was it run?
- what was the outcome?
recording the code that was run

› store a copy of the executable
› or of the source code
› including that of any libraries used
› as well as the compiler used
› and the compilation procedure
recording the code that was run

- the version of the interpreter
- and any options used in compiling it
- a copy of the simulation script
- and of any external modules or packages that are imported/included
recording the code that was run

maybe instead of storing a copy of the code we can store the repository URL and version number
recording the platform

- processor architecture
- operating system
- number of processors
What should this automated lab notebook look like?
different researchers, different workflows

➤ command-line

➤ GUI

➤ batch jobs

➤ solo or collaborative

➤ any combination of these for different components and phases of the project
Requirements

- Automate as much as possible, prompt the user for the rest
- Interact with version control systems (Subversion, Git, Mercurial, Bazaar)
- Support launching serial, distributed, batch simulations
- Link to data generated by the simulation
- Support all and any (command-line driven) simulation programs
- Support both local and networked storage of simulation information
Requirements

- Be very easy to use, or only the very conscientious will use it

- Sample in crucible exploded

- No sample
Sumatra

http://neuralensemble.org/trac/sumatra
Sumatra Simulation Management Tool
Sumatra

Nothing to do with Java
a Python package, `sumatra`, to enable automated recording of provenance information

can be used directly in your own code

or as the basis for interfaces
Current

➢ a command line interface, `smt`

➢ a web interface, `smtweb`

Future

➢ could be integrated into existing GUls (neuroConstruct, Topographica, nrngui)

➢ or new desktop/web-based GUls written from scratch
Dependencies

- Python bindings for your preferred version control system (pysvn, mercurial)
- NeuroTools.parameters
- Django (only needed for web interface)
smt

$ cd myproject
$ smt init MyProject
$ python main.py default.param
$ smt run --simulator=python --main=main.py default.param
```
$ smt list
default_20090930-174949
default_20090930-175111

$ smt list -l
--------------------------------------------------
Label      : default_20090930-174949
Reason     :
Outcome    :
Duration   : 0.0548920631409
Script     : MercurialRepository at /path/to/myproject
            rf9ab74313efe (main file is main.py)
Executable : Python (version: 2.6.2) at /usr/bin/python
Timestamp  : 2009-09-30 17:49:49.235772
Tags       :
            .
            .
            .
```
$ smt configure --simulator=python --main=main.py
$ smt run default.param
$ smt info
Simulation project
------------------
Name                : MyProject
Default executable  : Python (version: 2.6.2) at python
Default script      : MercurialRepository at /path/to/myproject
                      rf9ab74313efe (main file is main.py)
Default launch mode : serial
Data store          : ./Data
Record store        : Relational database record store using the Django ORM (database file=.smt/sim_records)
$ 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 comment default_20090930-174949 "Eureka! Nobel prize here we come."
$\text{smt tag "Figure 6"}$
$ smt run --reason="test effect of a smaller time constant" default.param tau_m=10.0
$ smt repeat haggling_2009101002
The simulation results match.
$ smt
Usage: smt <subcommand> [options] [args]

Simulation management tool, version 0.1

Available subcommands:
  init
  configure
  info
  run
  list
  delete
  comment
  tag
  repeat
$ smt comment --help

Usage: smt comment [options] [LABEL] [COMMENT]

This command is used to describe the outcome of the simulation. If LABEL is omitted, the comment will be added to the most recent simulation. If the '-f/--file' option is set, COMMENT should be the name of a file containing the comment, otherwise it should be a string of text.

Options:
- -h, --help show this help message and exit
- -r, --replace if this flag is set, any existing comment will be overwritten, otherwise, the new comment will be appended to the end, starting on a new line
- -f, --file interpret COMMENT as the path to a file containing the comment
Using sumatra within your own scripts

```python
import numpy
import sys

parameter_file = sys.argv[1]
parameters = {}
execfile(parameter_file, parameters)  # this way of reading parameters
# is not necessarily recommended

numpy.random.seed(parameters["seed"])
distr = getattr(numpy.random, parameters["distr"])
data = distr(size=parameters["n"])  

output_file = "example.dat"
numpy.savetxt(output_file, data)
```
import numpy
import sys
import time
from sumatra.projects import load_simulation_project
from sumatra.programs import Script
from sumatra.parameters import build_parameters

project = load_simulation_project()
start_time = time.time()

parameter_file = sys.argv[1]
parameters = build_parameters(parameter_file)

script = Script(main_file=__file__)
script.update_code()

sim_record = project.new_record(parameters=parameters,
                                 script=script,
                                 label="api_example",
                                 reason="reason for running this simulation")

numpy.random.seed(parameters["seed"])
distr = getattr(numpy.random, parameters["distr"])
data = distr(size=parameters["n"])

output_file = "%s.dat" % sim_record.label
numpy.savetxt(output_file, data)

sim_record.duration = time.time() - start_time
sim_record.data_key = sim_record.datastore.find_new_files(sim_record.timestamp)
project.add_record(sim_record)

project.save()
open-source
modular, extensible structure
contributions welcome
Coming soon

➤ support for MPI-based and for batch simulations

➤ improved recording of version information (versions of all imported Python modules, etc.)

➤ improved recording of platform information

➤ support for more parameter file formats

➤ improvements to the web interface

➤ remote record storage (for collaborative projects, etc.)
Coming later*

- better integration of post-simulation analysis
- desktop GUI application
- support for Git, Bazaar, etc.

*unless someone else would like to implement them sooner
Questions?

Sumatran orangutan by BelalangJantan http://www.flickr.com/photos/7164478@N07/3575735482/