Euroscipy 2011: Python in Neuroscience workshop

Paris, Ecole Normale Supérieure, August 29-30 2011

Program committee

- Bertrand Thirion (INRIA Saclay)
- Romain Brette (ENS Paris)
- Eilif Müller, Blue Brain Project, EPFL Laussane
- Gaël Varoquaux, INSERM U992, Saclay
- Raphaël Ritz, INCF, Stockholm, Sweden
- Laurent Perrinet, INCM, Marseille
- Andrew Davison, UNIC, CNRS, Gif

Sponsored by

- ITMO Neurosciences, Sciences cognitives, Neurologie, Psychiatrie
- Ecole Normale Supérieure
- INRIA Saclay-Île-de-France
- INCF-France

Python in Neuroscience satellite to Euroscipy

29-30 Aug 2011
France

Table of Contents

“neural simulation”

Title
=====
DANA, Distributed Asynchronous Numerical & Adaptive computing framework

Motivations
===========

Computational neuroscience is a vast domain of research going down from the very precise modeling of a single spiking neuron, taking into account ion channels and/or dendrites spatial geometry up to the modeling of very large assemblies of simplified neurons that are able to give account of complex cognitive functions. DANA attempts to address this latter modeling activity by offering a python computing framework for the design of large assemblies of neurons using numerical and distributed computations.

Implementation
==============

DANA is a python computing framework based on numpy and scipy libraries whose primary goals relate to computational neuroscience and artificial neural networks. However, this framework can be used in several different areas like physic simulations, cellular automata or image processing. The computational paradigm supporting the DANA framework is grounded on the notion of a unit that is a essentially a set of arbitrary values that can vary along time under the influence of other units and learning. Each unit can be connected to any other unit (including itself) using a weighted (possibly adaptive) connection. A group is a structured set of such homogeneous units.

For example the game of life can be written very simply as follow:

```
>>> from dana import *
>>> src = Group((100,100),
          '''V = maximum(0,1.0-(N<1.5)-(N>3.5)-(N<2.5)*(1-V)) : int
             N : float''')
>>> C = SparseConnection(src('V'), src('N'),
                         np.array([[1., 1., 1.],
                                   [1., 0., 1.],
                                   [1., 1., 1.]]))
```

src has been defined as a group of 100x100 units, each of them having a single value V whose value is computed according to V equation and where N designates a connection from Z to Z (using specified kernel in C definition). To simulate the game of life, one can write:

```
>>> src.V = rnd.randint(0, 2, src.shape)
>>> run(n=100)
```

In the example above, the connection has been made static, but it could have been made adaptive by defining an equation dW/dt for the C connection.

The DANA framework offers a set of core objects to design and run such models with the possibilty of specifying model equations (differentials/regular), connections and connection equations.

Finally, it is to be noted that DANA has been made purposely very similar to the BRIAN spiking neural networks simulator and attempt to re-use its syntax as much as possible.

More information at http://dana.loria.fr

A Python toolbox for fitting neuron models to electrophysiological data

Computational models of single neurons are increasingly being used in systems neuroscience to understand how function emerges in biological neuron networks. There are two broad categories of neuron models. Biophysical models, like the Hodgkin-Huxley model, describe the membrane potential dynamics through ion channels openings and closings, and were successfully used to understand action potential generation. Phenomenological spiking neuron models, such as integrate-and-fire neuron models, are based on simplified subthreshold voltage dynamics and a threshold-based instantaneous spike generation, and are mainly used when only output spikes are needed rather than the entire voltage trace. It has been recently shown that these phenomenological models can predict the response of cortical neurons to somatically injected currents with very good accuracy in spike timing. They are consequently widely used in network simulations, where only the input current/output spike trains relationship matters.

Although spiking models are now well established, finding relevant parameters with respect to biological neurons is not straightforward. Several methods have been developed but are generally not generic, which limits their practical applicability. This is a difficult optimization problem because 1) any matching criterion based on spike times is discontinuous, because of the fundamentally discontinuous nature of spike generation in these models, 2) the evaluation of a single parameter set involves the simulation of a model over a potentially very long time. There is for those reasons a need of a fitting procedure which is both generic and computationally efficient.

We have developed a model fitting toolbox[1] that lets the user provide a model through its equations in mathematical form, and computes the best parameters with respect to electrophysiological data specified by the user. It is based on the optimization of a matching criterion between the model and the data, such criterion being computed in an online fashion during model simulations. Both biophysical and spiking models are supported, through the use of spike-based or voltage-based matching criteria, whereas electrophysiological data can be provided as either spike trains or membrane potential traces.

The toolbox is written in Python and is provided as a built-in toolbox for the Brian simulator[2]. To achieve efficiency, it uses vectorization techniques and it can run on a graphics processing unit (GPU), an inexpensive yet very powerful chip available on most modern computers and containing hundreds of processor cores. The model fitting toolbox is also based on the Playdoh package[3], a pure Python parallel computing package making it possible to distribute heavy computations such as optimizations across multi-core computers connected inside a standard Ethernet network.

We demonstrate its use on intracellular recordings in the barrel cortex and show that we can obtain results with simple adaptive spiking models similar to those of the winning models in the 2009 INCF quantitative single neuron modeling competition[4]. We also show that our toolbox can be used to reduce a complex multicompartmental neuron model to a simple effective spiking model. Finally, we report that the toolbox can also be used to perform a calibration-free offline electrode compensation in intracellular recordings with high electrode resistance.

---

[1] http://www.briansimulator.org/docs/modelfitting.html
[2] http://www.briansimulator.org/
[3] http://code.google.com/p/playdoh/
[4] http://www.incf.org/community/competitions/spike-time-prediction/2009

Python Tools for Neuromorphic Systems Configuration

E. Neftci, F. Stefanini, S. Sheik, E. Chicca, G. Indiveri

June 7, 2011

In the past recent years several research groups have proposed neuromorphic Very Large Scale Integration (VLSI) devices that implement event-based sensors or bio-physically realistic networks of spiking neurons. It has been argued that these devices can be used to build event-based systems, for solving real-world applications in real-time, with efficiencies and robustness that cannot be achieved with conventional computing technologies.

In order to implement complex event-based neuromorphic systems it is necessary to interface the neuromorphic VLSI sensors and devices among each other, to robotic platforms, and to workstations (*e.g.* for data-logging and analysis). This apparently simple goal requires painstaking work that spans multiple levels of complexity and disciplines. Within this context, we developed a framework named pyNCS (python Neuromorphic Cognitive Systems) to simplify the configuration of multi-chip neuromorphic VLSI systems and the definitions of network architectures [1].

In pyNCS, emphasis is given on modularity and applicability to general multi-chip systems. A set of XML files describes the neuromorphic setup and the chips, which greatly simplifies the specification of neuromorphic setups of any kind.

The pyNCS framework is developed entirely in Python, and features practical address specification definition (for the translation between raw (hardware) and neural addresses); abstraction of neurmorphic neural populations and connections by dedicated software classes; spiketrain analysis and plotting (via NeuroTools); real-time chip activity monitoring; communication of hardware events over a network interface; systematic tuning of the network parameters and inter-operability with pyNN (in progress).

References

[1] Sadique Sheik, Fabio Stefanini, Emre Neftci, Elisabetta Chicca, and Giacomo Indiveri. Systematic configuration and automatic tuning of neuromorphic system. In *International Symposium on Circuits and Systems, ISCAS 2011*. IEEE, 2011. submitted.

PyNN: a unified interface for neuronal network simulators

Andrew Davison[1], Eilif Muller[2], Mike Hull[3], Pierre Yger[1].

[1]Unité de Neuroscience, Information et Complexité, CNRS UPR 3293, Gif-sur-Yvette, France
[3]Blue Brain Project, EPFL, Lausanne, Switzerland
[3]University of Edinburgh, UK

Computational neuroscience has produced a diversity of simulator software for simulations of networks of spiking neurons. This diversity has both positive and negative consequences. The principal problem is that each simulator uses its own programming or configuration language, leading to considerable difficulty in porting models from one simulator to another or even in understanding someone else's code. This impedes communication between investigators and makes it harder to reproduce or build on other people's work. The advantages of having multiple simulators available are (i) each simulator has a different domain of excellence, allowing the most appropriate software to be chosen for a given problem; (ii) simulation results can be cross-checked between different simulators, giving greater confidence in their correctness.

The Python package PyNN provides a unified API across neural simulators (currently NEURON, NEST, PCSIM, Brian and a neuromorphic VLSI hardware implementation, with support for MOOSE and GENESIS 3 under development), making it possible to write a simulation script once and run it without modification on any supported simulator or hardware platform.Thus we keep the advantages of having multiple simulators (for cross-validation, etc) but remove the translation barrier.

Here we present recent developments within PyNN, notably improved support for higher-level network structures, support for the NineML and NeuroML languages for describing neuronal and synapse models, and support for the Connection Set Algebra for describing network connectivity.

PyNN is open-source software and is available from http://neuralensemble.org/PyNN.

Title: CLONES: A Closed-Loop Simulation Framework for Body, Muscles and Neurons

Abstract :

The activity of neurons does not only reflect cognitive states, it is also highly constrained by mechanical properties of the body: sensors, muscles, tendons, and their interaction with the environment. Therefore, in order to model the activity of neurons, it seems necessary to take a holistic approach, where the behavior of an animal is modeled through the interaction between neurons, muscles and the environment. This involves knowledge in both neurophysiology and biomechanics. It also requires the development of software that can simulate the interaction between neurons and muscles efficiently. In order to achieve this, we have developed an open-source library called CLONES (Closed Loop Neural Simulation).

CLONES is a communication interface between the BRIAN neural simulator, and SOFA, a physics engine for biomedical applications. BRIAN and SOFA are both intuitive and high performance simulation environments. BRIAN is based on the Python programming language. SOFA uses an interpreted XML description of a physical scene. CLONES contains a SOFA plugin and a PYTHON module. The Python module provides a function to be called after each step of the simulation. The Sofa plugin provides components for the simulation of muscles, sensors, drag forces. Communication between BRIAN and SOFA is achieved through shared memory and semaphores. A single step of simulation typically takes place on different CPU cores. Once a simulation step has been completed in both simulators, sensory inputs to the neurons and motor outputs to the muscles are updated. Several demonstration examples are provided with the library. In particular, a neuro-mechanical model of undulatory locomotion in the worm *caenorhabditis elegans* is available.

"What's new with Brian?"

Speakers: Romain Brette, Cyrille Rossant, Victor Benichoux

Abstract:
We will give a brief overview of the Brian simulator (http://briansimulator.org) and present the most recent additions, in particular Brian Hears, a toolbox to simulate auditory models, and the model fitting toolbox, to fit arbitrary spiking models to electrophysiological data. We will also present planned developments, such as GPU simulation.

“electrophysiology”

NeurOnline: A software to perform online analysis and control of electrophysiological recordings

Maxime Ambard, Armin Brandt and Stefan Rotter
Bernstein Center Freiburg & Faculty of Biology, University of Freiburg, Germany

Project Summary

It is now standard to record the activity from large numbers of neurons simultaneously, both in behaving animals using acute or chronically implanted electrode arrays, and in brain slices or tissue cultures using substrate-integrated multi-electrode arrays. To enable control and intervention in an ongoing experiment, it is important to monitor certain critical parameters of the recorded activity in real-time. NeurOnline is a software that enables the researcher to perform online analysis of their electrophysiological recordings and to suitably interact with their experimental setups based on these analysis results. Specifically, our software supports optimizing the yield of experiments by providing new algorithms for online analysis that give comprehensive feedback about the status of the experiment in real-time. In addition, we hope to stimulate the development of novel experiments based e.g. on the possibility of fast adaptation of applied stimuli depending on the behavior of the system studied. The software is made publicly available under GPL.

Technical details

The software architecture chosen for this project consists in Python scripts that call C++ extensions provided by SIP. The C++ language allows high-performance computations that are crucial for time-critical "online" analyses and an easy use of multiple threads. The Python scripting language, on the other hand, enables experts and semi-skilled programmers at the same time to easily use and extend the envisaged software toolbox. The QT library is used to implement a signal/slot mechanism. A graphical user interface permits to control the analysis and displays the results of all computations performed on the recorded data.

Based on a set of open source drivers ("comedi"), NeurOnline can currently interact with 400 different data acquisition boards (e.g. National Instruments) that are commonly employed in electrophysiological setups. A TCP/IP client has been implemented to allow the communication with high-density multi-electrode arrays (HD-MEAs) currently developed at ETH Zurich/Basel. These devices currently allow sampling from 128 channels (selected from a set of 11,000) at a sampling frequency of 20 kHz per channel. Acquisition from Multichannel Systems multi-electrode arrays is also supported.

A butterworth IIR online filter has been developed to select the appropriate frequency bands of the recorded signals. Online spike sorting is performed on the detected spike waveforms by applying a dynamic template matching algorithm. The Python interface allows NeurOnline to send signals that depend on the result of online signal analysis to other processes, e.g. to update in real-time the visual stimulus displayed by some dedicated software (e.g. "VisionEgg").

NeurOnline is currently used in two laboratories: At the Biomicrotechnology laboratory (IMTEK, University of Freiburg) the dynamics of dissociated cell cultures grown on HD-MEAs are studied. Those arrays have 11,000 recording sites, 128 of which can be recorded at the same time. NeurOnline is currently used to record the data from subsets of electrodes, to detect the spikes in the recorded signals and to organize the scan of the full set of electrodes depending on the recorded activity. At the Neurobiology and Biophysics laboratory (Faculty of Biology, University of Freiburg), extra- and intra-cellular recordings in the neocortex of anesthetized rats are performed. Visual stimuli that preferentially activate neurons in the thalamus (LGN) and in the primary visual cortex (V1) are selected by a method known as adaptive sampling. NeurOnline used in this laboratory for multi-channel signal recording, spike detection, spike sorting and visual stimulus updating.

Funding by the German Ministry of Education and Research (Bernstein Focus Neurotechnology Freiburg*Tübingen, FKZ 01 GQ 0830) is gratefully acknowledged.

Using Python tools for analysis of a mathematical model of after-depolarisation

Jakub Nowacki*, Hinke M. Osinga, Krasimira Tsaneva-Atanasova

Bristol Centre for Applied Nonlinear Mathematics,
Department of Engineering Mathematics,
University of Bristol, Queen's Building, University Walk, BS8 1TR

Abstract

Python combines usability, clarity and extensibility that makes it a language of choice for many projects concerning scientific computations. One of the unique feature of Python is the simplicity of interfacing between different programming languages. This feature have brought Python effectiveness into many established numerical tools, that previously were not necessary user friendly or constrained by the original language limitations. An examples of such a tool is the numerical continuation software AUTO [1]. The recently added Python interface considerably increased its usability and improved the data handling. These changes have created opportunities for the software to be used for various new applications. We present an example of a novel Python/AUTO application to perform an analysis of our model of after-depolarisation.

Understanding the behaviour of a biological model is a fundamental question from both theoretical and experimental point of view. Parameter uncertainty is a common feature of biological systems and therefore exploring the parameter space of a model could often lead to useful predictions. Currently, the vast majority of parameter sensitivity analysis is performed using brute force numerical simulation that could produce inherently incomplete results for example in the case of multi-stability. We propose a novel approach to parameter sensitivity analysis that employs numerical bifurcation theory techniques such as boundary value problem (BVP) formulation.

Our formulation allows us to use numerical continuation tools, such as AUTO, to explore the behaviour of a model under a parameter variation in a nearly continuous way. The continuation method enables us to determine regions in the parameter space where certain model behaviour of interest occurs. Using the BVP formulation we analyse a simplified model of after-depolarisation based on previous studies of a more detailed model of intrinsic excitability of CA1/3 pyramidal neurons [4]. Using the well-posed two point BVP continuation we identify the onset of the ADP and a burst in a form that allows us to perform a two-parameter continuation. This in turn enables us to unambiguously determine the boundaries (in parameter space) of the ADP and bursts with different number of action potentials. To confirm the continuation results we use XPPy [3], a Python interface for XPPAut [2], to perform model simulations. The visualisation of the results of continuation and simulations is done using Matplotlib. Finally our results provide useful predictions for further studies of the intrinsic excitability of pyramidal neurons.

References

1. E J Doedel and B E Oldeman. AUTO-07P: Continuation and bifurcation software for ordinary differential equations. http://cmvl.cs.concordia.ca/auto/, 2009.

2. Bard Ermentrout. *Simulating, analyzing, and animating dynamical systems: a guide to XPPAUT researchers and students*. SIAM, 2002.

3. Jakub Nowacki. XPPy. http://seis.bris.ac.uk/~enxjn/, 2010.

4. Jakub Nowacki, Hinke M. Osinga, Jon T Brown, Andrew D Randall, and Krasimira Tsaneva-Atanasova. A unified model of CA1/3 pyramidal cells: An investigation into excitability. *Progress in biophysics and molecular biology*, 105(1-2):34–48, March 2011.

*Corresponding author: `j.nowacki@bristol.ac.uk`

Optimisation of stimulation patterns for specific questions in electrophysiology experiments : a Python framework

Damien Drix, Thomas Nowotny

June 8, 2011

In order to investigate the properties and function of electrically active membranes, in particular neurons, scientists use electro-physiological recordings in two established experimental modes : voltage-clamp and current-clamp. In both cases, a variety of stimulation patterns is used to elicit salient responses. The stimulation patterns are typically generated via a small number of established methods. The most common voltage clamp protocols use a combination of square steps and pre-steps to characterise the ionic conductances in a membrane ; in other cases sinusoids and noise sources have been used. The choice of a particular stimulus depends on the aim of the experiment ; for instance, one can select the amplitude and duration of voltage steps so as to highlight the effect of a potassium conductance, or apply high-frequency sinusoidal current injections to determine the membrane capacitance. Similarly, white noise has been used in model fitting tasks.

Here we propose a set of generic methods for systematically generating stimuli tailored to specific tasks, implemented within a simulated electrophysiology framework in Python. Our approach involves the use of splines to represent stimulation patterns, and genetic algorithms to evolve patterns that are particularly suitable for an arbitrary given task. Splines permit defining a curve through a number of given support points. By manipulating the positions of the support points and the type of spline interpolation, steps and sinusoids can be approximated, as well as other arbitrary shapes. This versatile encoding allows the optimisation algorithm to exploit any pattern that yields the desired result.

We developed a Python framework called `evopy` to handle the evolutionary search. This framework is able to leverage Enthought's Traits attributes to generate a real-valued or binary genetic encoding for arbitrary objects. Thus, the allowed parameter ranges and their inclusion or exclusion in the search can be easily controlled via metadata, with any change in the object model seamlessly reflected in the genetic encoding, encouraging experimentation. The optimisation itself is performed with a multi-objective genetic algorithm. Particle Swarm Optimisation and an interface to `scipy.optimize` routines are also available.

Since speed is a concern with meta-heuristic optimisation methods, we use a mixture of Python code to manage the experiments and analysis together with C and Cython components for critical sections. We define our Hodgkin-Huxley neuron models and simulated electrophysiology components as Cython classes. Similarly, ODE solvers from the GNU GSL library and spline interpolation code from the ALGLIB library are used through Cython language bindings. Thus, the entire simulation runs in native code, avoiding costly Python calls during integration.

We demonstrate the use of this framework to generate stimuli optimised to isolate the contributions of various Hodgkin-Huxley parameters to the dynamics of a model neuron in voltage- and current-clamp. Future directions using active stimuli in a dynamic clamp setup are also presented.

Flexible spike sorting in Python

Bartosz Telenczuk 1,2, Dmytro Belevtsoff 2,3

1 Insitute for Theoretical Biology, Humboldt University, Germany, Berlin
2 Neurology, Medical Uninversity Charite, Germany, Berlin
3 Bernstein Center for Computation Neuroscience, Germany, Berlin

Spike sorting is a common pre-processing step in the analysis of single or multi-unit activity (SUA or MUA). The goal of the procedure is to detect the times at which a single cell generated action potentials based on the extracellular recordings of an electric potential close to the cell. Since multiple cells can be active simultaneously special methods must be used to discriminate the responses of just a single cell. In many situations this is possible, because activities of different cells usually differ in wave shapes. Therefore it is possible to isolate them by comparing the shapes and amplitudes of detected extracellular spikes and grouping the ones which look similar (using automatic or manual clustering procedure).

There are dozens of different (commercial and free) software packages aimed at spike sorting. However, many of them are controlled only via graphical user interface (GUI) making them very inflexible. Testing new algorithms or adding support for new data formats usually requires in depth knowledge of the source code of the package (if available) and time-consuming development of extensions (if possible).

Therefore, we developed a new spike sorting library based on dynamic and interactive language (Python) called SpikeSort. While still very early in development, it offers many standard and not-so-standard algorithms for spike detection, feature extraction and spike classification. The main design goals of the library is the flexibility and extendibility allowing user to add new filters/algorithms/etc. without need to recompile or to understand the entire code base. In addition, the algorithms available in SpikeSort may be easily used in own scripts and programs.

We put much effort to make working with SpikeSort very interactive and intuitive. To this end, we employ a modular approach based on a set of components that may be flexibly combined to develop customized processing workflows.

Last but not least, we also take care of memory efficiency and performance. This is achieved among others by leveraging a set of third-party Python libraries (NumPy, PyTables and scikits.learn).

“data management and databasing”

Lessons learned from neuroimaging tool development in Python

Stephan Gerhard

May 30, 2011

The large complexity of the systems neuroscientific researchers aim to investigate demands sophisticated infrastructure and technology. To tackle the challenges associated with these technical developments, the field of neuroinformatics is emerging.

For a large variety of problems in neuroscience and their associated computational requirements, the Python programming language has been adopted for tool and library development.

Python thus may play a central role in the evolution of the developments necessary to tacke the upcoming technical challenges neuroscience researchers are faced with. Python is apt to be the glue for different legacy systems, and to build tools and libraries to support research questions in a straightforward fashion.

In my talk, I would like to present my subjective experience on Python tool development for the Diffusion MRI community. A number of technical and non-technical aspects of the process will be highlighted, a number of questions are raised which are important to be considered early on in the developmental process and a number of decisions will be outlined which I took in my own projects which from the more experienced perspective I share now, could have been taken differently.

The information I would like to convey aims to serve other researchers starting with Python development to arrive at improved decisions pertaining their own projects. Five relevant aspects shall be touched upon:

Research Environment & Community Reflect on your commitment: Is it tool development à la "Getting-The-Job-Done-For-Me" vs. a community-targeted, reusable, documented, tested tool with a stable API? Do you see a chance of building a community around your tools? Does a community already exist where you could possibly contribute? Do you have close collaborates with a similar vision that care for the tools your are going to develop? What are the constraints of your supervisors and your institution? How can you overcome concerns of making the code available as open source? What license to choose? Who will own the copyright? What are the current practices and tools employed in your lab, and are you able and willing to educate people in order to change them?

The User Perspective How to distinguish between mere users (that do not care about software design or methodological concerns), and user-developers (doing method development and applied research), methods people (focusing on the improvement or creation of new methods, the user interface is very secondary). How to best approach different groups adequately? What are the advantages and disadvantages of building a GUI on top of libraries? Similarly, user-Dialogs vs. writing a scripts? Do you want your code to be used at all by others and can you live with the implications thereof? How can you overcome obstacles of proper software distribution and multi-platform support? What happens if you do not develop according to user requirements, but according to imagined user requirements?

Recommended supporting tools How to adapt to your particular research environment? What are the habits of your collaborators? Do you have ideas to change their habits for the better? What IDE and code editors would you recommend to lab mates? Why to encourage to write unit tests? How to write documentation fast and properly? How to interact with your users?

My tool contributions (so far) I will give a short overview of the tools I have been working on for the last two and a half year, and how they evolved. How modularization and working on proper interfaces turned out to be very beneficial. Why writing things up in a paper is important, and which journals accept software submissions. Why sometimes reinventing the wheel is good, and when to contribute to existing open source projects?

Open Science What would be a vision for "Python in Neuroscience", and how would it tie into the emerging open science movement? Why do we need reusable, well-documented and tested libraries, especially in neuroscience? What could be your contribution?

NiBabel: Conductor for a cacophony of neuro-imaging file formats

Matthew Brett[1], Stephan Gerhard[2], and Michael Hanke[3]

[1] *Helen Wills Neuroscience Institute, University of California at Berkeley, USA*
[2] *Electrical Engineering, Ecole Polytechnique Fédérale de Lausanne, Switzerland*
[3] *Center for Cognitive Neuroscience, Dartmouth College, Hanover, New Hampshire, USA*

Neuro-imaging research has always involved spending countless hours on getting access to data in an intoxicating variety of formats. While the neuro-imaging community attempts to converge on a reasonable subset of the available formats for the purpose of data exchange, *convenient* data access remains a challenge that affects interoperability of software and perceived joy of research collaborations.

To fully exploit the power of a growing manifold of Python-based software for neuro-imaging research it is essential to have an easy-to-install, easy-to-use system that facilitates access to data in any necessary format. *NiBabel*[1] aims to be this basic IO layer for neuro-imaging data format access in Python. While other solutions have been around for several years (e.g., PyNIfTI), they were limited in their scope, the variety of supported data formats, or had portability issues that made their deployment unnecessarily difficult.

This presentation introduces the general architecture of the pure-Python package NiBabel that accents on the concept of making common tasks simple, and difficult tasks possible. It will illustrate NiBabel's basic image model and show how it can be tailored to represent images in various flavors of ANALYZE, GIFTI, NIfTI1, MINC, and PAR/REC – without loosing access to information embedded in the meta data of the respective formats. Moreover, the presentation will outline how developers can contribute functionality, and use NiBabel as an open platform for general purpose neuro-imaging data format access that is available to any Python user or developer – without licensing restrictions and minimal dependencies.

[1] http://nipy.sourceforge.net/nibabel

Neo: representing and manipulating electrophysiology data in Python

Andrew Davison[1], Thierry Brizzi[1], Luc Estabanez[1], Florent Jaillet[2], Yann Mahnoun[3], Philipp Rautenberg[4], Andrey Sobolev[4], Thomas Wachtler[4], Pierre Yger[1] and Samuel Garcia[5]

[1]Unité de Neuroscience, Information et Complexité, CNRS UPR 3293, Gif-sur-Yvette, France
[2]Institut de Neurosciences Cognitives de la Méditerranée, CNRS UMR 6193 - Université de la Méditerranée, Marseille, France
[3]Laboratoire de Neurosciences Intégratives et Adaptatives, CNRS UMR 6149 - Université de Provence, Marseille, France
[4]G-Node, Ludwig-Maximilians-Universität, Munich, Germany
[5]Laboratoire Neurosciences Sensorielles, Comportement, et Cognition, CNRS UMR 5020 - Université Claude Bernard, Lyon, France

In neuroscience, electrophysiological signals, whether from electroencephalography, intracellular and extracellular electrophysiology, or simulation, are acquired, analysed and visualized using a wide variety of software, much of it proprietary, developed by recording hardware manufacturers, very often written in Matlab, and increasingly written in Python.

To improve interoperability of different tools and to share data between different projects, a common representation of the core data is needed. A number of efforts have been made in this direction, including the Neuroshare project (http://neuroshare.sourceforge.net) for proprietary formats, and the FIND (http://find.bccn.uni-freiburg.de/) and sigTOOL (http://sigtool.sourceforge.net/) toolboxes for Matlab. For Python, however, such a common representation has so far been missing.

We have developed a Python package, Neo, for representing electrophysiology data in memory, and for reading/writing the data to/from a variety of commonly-used file formats. Neo is deliberately limited to represention of data, with no functions for data analysis or visualization, in order to be as lightweight a dependency as possible. A common package or packages for neurophysiology data analysis and visualization, such as the NeuroTools package (http://neuralensemble.org/NeuroTools), can then build on top of Neo. Currently Neo is used as the core representation of data in OpenElectrophy (http://neuralensemble.org/trac/OpenElectrophy) and it is planned to also use it in NeuroTools and in the G-Node portal (http://www.g-node.org/) in the near future.

Neo implements a hierarchical data model well adapted to intracellular and extracellular electrophysiology and EEG data with support for multi-electrodes (for example tetrodes), including classes such as SpikeTrain and AnalogSignal. The Neo package also provides a set of input/output (IO) modules for various neurophysiology file formats (Plexon, Spike2, NeuroExplorer, Axon, AlphaOmega, Micromed, EEGLab, WinWcp, Elan, Elphy, PyNN, Neuroshare (Win32 only), TDT, ASCII and raw binary data). Neo builds on the quantities package (http://pypi.python.org/pypi/quantities), which in turn builds on NumPy.

Neo is distributed under a BSD licence, and is available through the Python Package Index (http://pypi.python.org/pypi/neo/), with documentation at http://packages.python.org/neo/ and a development website at http://neuralensemble.org/trac/neo. New contributions are welcome.

“neuroimaging data processing”

BrainVISA: a complete software platform for neuroimaging

D Geffroy[1], D Rivière[1,2], I Denghien[1], N Souedet[1,3], S Laguitton[1,2], Y Cointepas[1,2]

[1]IFR 49, Gif-sur-Yvette, France/[2]CEA, I2BM, Neurospin, Gif-sur-Yvette, France/[3]CEA, I2BM, MIRCEN, Fontenay-aux-Roses, France

BrainVISA is a modular and customizable software platform built to host heterogeneous tools dedicated to neuroimaging research. It aims at helping researchers in developing new neuroimaging tools, sharing data and distributing their software. It is developped by research organizations (mainly CEA, INSERM, INRIA, CNRS) grouped in a federative research institute (IFR 49) founded by the french government. It is a free and open-source software written in Python language and can be downloaded from: **http://brainvisa.info.**

BrainVISA offers numerous features in several domains of neuroimaging software:

Data management

As most neuroimaging software deal with neuroimaging data as files, BrainVISA keeps data files on the filesystem, and is using a database to index them. BrainVISA database system is based on a customizable **ontology** that defines the data types handled by the software, associated key attributes for indexation, and filename patterns to make the link between the filesystem and the database schema. There is a bijection between the files organization and the database which makes it possible to get an input or output file name from data attributes values through a simple database request.

BrainVISA currently uses **SQLite** as a SQL database engine because it is simple, easy to use and does not need any server. Moreover, SQLite can be directly used from the Python language. The database is stored in a single file that can be copied from one computer to another, making sharing databases easier for users.

Workflows and pipelines

BrainVISA offers several neuroimaging processing pipelines. A pipeline is a combination of reusable executable components called *processes*, it groups together several processing steps.

Developping new pipelines for BrainVISA is a matter of writing small scripts in Pyhton (Fig. 1). They may jointly use home-made algorithms, software contained within the core BrainVISA package, or third-party software. Some existing BrainVISA pipelines use for example SPM, FSL, Freesurfer, etc.

Graphical user interface

BrainVISA graphical user interface has been developped with PyQt, the python bindings of the Qt library. A view is automatically generated for each pipeline to allow the user to fill in the parameters. This view is created from a simple python description of the process input and output parameters. It also shows a panel to monitor the execution of the pipeline. Of course, it is possible to customize the default user interface if it does not fit the user needs.

BrainVISA may also work as batch and without graphical interface, to run a specific process or pipeline. This thus allows for instance, batch scripting or parallel distribution, and the use of BrainVISA components from custom software.

Visualization

BrainVISA offers a way to define viewers which may use any visualization software. A viewer is a processes dedicated to the visualization of data of a specific type. Most of BrainVISA existing viewers use **Anatomist,** a powerful software for 3D visualization and manipulation of structured objects. Anatomist, controled through its python API by BrainVISA, provides an interactive visualization, suitable for any kind of data.

Massive computation facilities

In order to ease the processing of large neuroimaging databases, BrainVISA provides a feature to **iterate** a process or a pipeline on whole or part of a database. Input data are selected through a request to the database and all generated results will be written automatically at a suitable location on the filesystem and indexed in the database thanks to the ontology.

Since the 4.1 version, it is also possible to submit BrainVISA pipelines or iterated processes to parallel computing ressources thanks to the **Soma-workflow** software. Soma-workflow is an independent software component of the BrainVISA environment, which provides a homogeneous and single way to submit and monitor parallel computing to various computing resources (laptops, workstation farms, clusters...). See its home page on http://brainvisa.info/soma/soma-workflow/ for more information.

Toolboxes

As BrainVISA is designed to be extensible, it is possible to create a custom toolbox which includes a set of processing tools, an ontology definition, documentation, and any needed software and data. Toolboxes may be distributed as separate add-on packages.

Thus, BrainVISA infrastructure advantages have encouraged several teams to develop their own toolboxes. A lot of neuroimaging tools are already available in **BrainVISA toolboxes**. For example, it is possible to perform automatic segmentation of brain hemispheres and sulci from T1 MR images with **Morphologist** toolbox, automatic sulci identification and object-based morphometry with **Sulci** toolbox, cortical surface geometry analysis and functional data projection with **Cortical Surface** toolbox, diffusion MR images analysis and fiber tracking with **Connectomist**, automated processing of

histological and autoradiographic sections with **BrainRat**, functional MRI data analysis with **fMRI** toolbox, etc.

Conclusion

BrainVISA has now many years of maturity and has been successfully used by various research groups. See http://brainvisa.info/biblio/en/index.html for a list of publications involving BrainVISA. BrainVISA has been chosen by several research centers and collaborative projects as the backbone infrastructure for data management, neuroimaging tools development and software distribution.

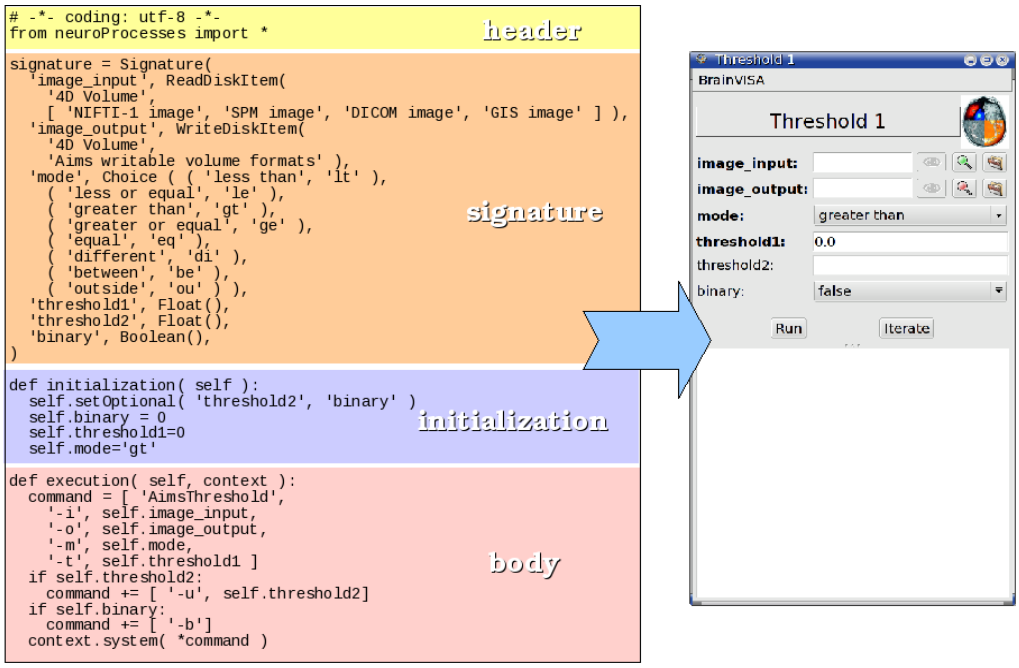


Fig. 1: A simple BrainVISA process: source code and default GUI

Classification, Induction and Brain Decoding

Emanuele Olivetti*†, Susanne Greiner*†

Email: olivetti@fbk.eu, greiner@fbk.eu
*NeuroInformatics Laboratory (NILab), Bruno Kessler Foundation, Trento, Italy
†Center for Mind and Brain Sciences (CIMeC), University of Trento, Italy

I. Abstract

Machine Learning (ML) techniques are provoking wide interest in brain-computer interface (BCI) research and neuroimaging-based neuroscience research. These techniques are most frequently applied according to a paradigm called *brain decoding* [1], which is based on the prediction of stimuli provided to a subject from the concurrent brain activity.

In [1] three specific tasks are recognized as part of the brain decoding approach. *Pattern discrimination* is the task that addresses the question whether and how accurately the brain activity can predict the triggering stimulus. This task is mainly focused on the quantitative assessment of the classification error rate. Means for interpretation of how the classification algorithm exploited the available information, e.g. brain maps, are not directly provided. Pattern discrimination is of special interest for BCI research and neuroscience research for different reasons: while the first is mainly concerned with getting the most accurate predictions based on the brain activity, the latter uses pattern discrimination for inductive reasoning and frequently for comparing different theories/hypotheses in the light of the evidence provided by the experiment, i.e. for confirmatory data analysis. In this case the primary interest is the reliability of the error estimate of the predictor, which can be in terms of a confidence interval or posterior probability.

To the best of our knowledge the use of classification algorithms for inductive reasoning in scientific research is marginal to the mainstream ML literature. Moreover the reliability of the strategies proposed in the neuroscience-related literature [1], [2] is often difficult to assess [3], especially from small high-dimensional samples typical of this domain. This work wants to raise the attention on the use of classification algorithms for inductive reasoning and proposes the application of the Bayesian hierarchical modeling and the Bayesian hypothesis testing (BHT) framework to the analysis of classification results. In order to support the proposed approach we present two recently-published applications:

1) A model and a test about the *information* between stimuli and brain data in single subject studies. See [4].
2) A multi-subject model and test to make inferences about the population of interest from the result of a brain decoding study on a group of subjects. See [5].

Our implementation of these two applications has been carried out in Python language on top of NumPy and SciPy, the Python software stack for scientific applications. This software stack proved to be extremely efficient both from the software development point of view and for computational aspects. In particular the broadcasting features of the NumPy `ndarray` are fully supported by the random sampling routines in the `numpy.random` subpackage allowing an extremely concise code that fully exploits the speed of C code.

The implementations of the two applications [1] [2] are available under a Free Software / Open Source license. Their integration within PyMVPA [3], the Python package for statistical learning analyses tailored to the neuroimaging domain, is under discussion.

References

[1] F. Pereira, T. Mitchell, and M. Botvinick, "Machine learning classifiers and fMRI: A tutorial overview," *NeuroImage*, vol. 45, no. 1, pp. 199–209, Mar. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.neuroimage.2008.11.007

[2] J. A. Etzel, V. Gazzola, and C. Keysers, "An introduction to anatomical ROI-based fMRI classification analysis," *Brain Research*, vol. 1282, pp. 114–125, Jul. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.brainres.2009.05.090

[3] A. Isaksson, M. Wallman, H. Goransson, and M. Gustafsson, "Cross-validation and bootstrapping are unreliable in small sample classification," *Pattern Recognition Letters*, vol. 29, no. 14, pp. 1960–1965, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2008.06.018

[4] E. Olivetti, S. Veeramachaneni, and P. Avesani, "Testing for Information with Brain Decoding," in *IEEE International Workshop on Pattern Recognition in NeuroImaging*, May 2011.

[5] E. Olivetti, S. Veeramachaneni, and E. Nowakowska, "Bayesian hypothesis testing for pattern discrimination in brain decoding," *Pattern Recognition*, May 2011. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2011.04.025

[1] http://github.com/emanuele/information_test
[2] http://github.com/emanuele/Bayes-factor-multi-subject.
[3] http://www.pymvpa.org

The virtues and sins of PyMVPA

Yaroslav O. Halchenko[1,2] and Michael Hanke[1,2,3]

[1] *Center for Cognitive Neuroscience, Dartmouth College, Hanover, New Hampshire, USA*
[2] *Department of Psychological and Brain Sciences, Dartmouth College, Hanover, New Hampshire, USA*
[3] *Department of Experimental Psychology, University of Magdeburg, Magdeburg, Germany*

Encouraged by a rise of reciprocal interest between the machine learning and neuroscience communities, multiple studies have demonstrated the superior explanatory power of statistical learning techniques for the analysis of neural data of different acquisition modalities, such as fMRI and EEG. While development and underlying theory of the machine learning methods require substantial skills in computer sciences and statistics, there are fortunately a large number of software packages readily available that implement various promising algorithms. However, these packages are typically implemented in either a very generic fashion or tuned to address a specific problem, with almost none of them geared towards neuroscientific data analysis, thus adoption of those methods is often impaired. To equip community with a specialized platform exposing developments in machine learning to neuroscience, we have developed PyMVPA[1] – Python Multivariate Pattern Analysis[2].

Expressive power of the Python language coupled with a variety of the core libraries for numerical computation in Python or interfaced from other computing platforms (e.g. R through RPy[3]) provide a solid foundation for PyMVPA. Modular design of PyMVPA makes it possible to express complex analysis workflows of neural data in just a few lines of code neither sacrificing readability nor flexibility, while interfacing typical analysis procedures to numerous underlying Python libraries without explicit user awareness of their different API.

To make PyMVPA robust and accessible for neuroscientists, we had to address various demands, from user interface adequacy to the convenience of deployment. This talk will overview the design and capabilities of the framework, and then will accent on particular aspects which could be of an interest to the Python scientific developers: distributions awareness, external dependencies handling, compatibility assurance, data encapsulation, excessive testing, custom logging, and basic literate programming. The talk will conclude with a roadmap of future work planned for PyMVPA development.

[1] M. Hanke, Y. O. Halchenko, P. B. Sederberg, S. J. Hanson, J. V. Haxby, and S. Pollmann. PyMVPA: A Python toolbox for multivariate pattern analysis of fMRI data. Neuroinformatics, 7(1):37–53, Mar. 2009. http://dx.doi.org/10.1007/s12021-008-9041-y

[2] http://www.pymvpa.org

[3] http://rpy.sourceforge.net/rpy2.html

Pyhrf: a package to extract and study hemodynamics from fMRI data

Thomas Vincent[1], Philippe Ciuciu[1], Lotfi Chaari[2], and Solveig Badillo[1]

[1]CEA/DSV/I[2]BM/Neurospin, LNAO, Gif-Sur-Yvette, France
[2]INRIA, MISTIS, Grenoble, France

Pyhrf is a set of tools for within-subject fMRI data analysis, focused on the caracterisation of the hemodynamics. More precisely, this software addresses the two main tasks involved in fMRI analysis: (i) detect and localize which cerebral regions are activated by a given experimental paradigm and (ii) estimate the underlying dynamics of activation by recovering the so-called Hemodynamic Response Function (HRF). The main outputs are then a set of 3D statistical maps of cerebral activations along with the time-series describing the HRFs for the set of all brain regions (4D volume). The analysis can also be performed on the cortical surface from projected BOLD signals and then produces functional textures to be displayed on the input cortical mesh. To this end, pyhrf implements two different approaches: a voxel-wise and condition-wise HRF estimation [1] and a parcel-wise spatially adaptive joint detection-estimation algorithm [2,3]. This tool provides interesting perspectives so as to understand the differences in the HRFs of different populations (infants, children, adults, patients ...). Within the treatment pipeline of an fMRI analysis, `pyhrf` steps in after data preprocessings (slice-timing, realignment, normalization).

`pyhrf` is mainly written in Python, with some C-extension that handle computation intensive parts of the algorithms. The package relies on classical scientific libraries: `numpy`, `scipy`, `matplotlib` as well as `Nibabel` to handle input/outputs and `NiPy` which provides tools for functional data analysis. `pyhrf` can be used in a stand-alone fashion and provides a set of simple commands in a modular fashion. The setup process is handled through `XML` files which can be adapted by the user from a set of templates. This format was chosen for its hierarchical organisation which suits the nested nature of the algorithm parametrisations. A dedicated `XML` editor is provided with a `PyQt` graphical interface for a quicker edition and also a better review of the treatment parameters. When such an `XML` setup file is generated *ab initio*, it defines a default analysis which involves a small real data set shipped with the package. This allows for a quick testing of the algorithm and is also used for demonstration purpose. In the same respect, an articifial fMRI data generator is provided where the user can test the behaviour of the algorithms with different activation configurations, HRF shapes, nuisance types (noise, low frequency drifts) and paradigms (slow/fast event-related or bloc designs). Concerning the analysis process, which can be computation intensive, `pyhrf` handles parallel computing through the python software `soma-workflow` for the exploitation of cluster units as well as multiple cores computers. Finally, results can be browsed by a dedicated viewer based on `PyQt` and `matplotib` which handles n-dimensionnal images and provide suitable features for the exploration of whole brain hymodynamical results.

References

[1] P. Ciuciu, J.-B. Poline, G. Marrelec, J. Idier, Ch. Pallier, and H. Benali, "Unsupervised robust non-parametric estimation of the hemodynamic response function for any fMRI experiment," *IEEE Trans. Med. Imag.*, vol. 22, no. 10, pp. 1235–1251, oct. 2003.

[2] T. Vincent, L. Risser, J. Idier, and P. Ciuciu, "Spatially adaptive mixture modelling for analysis of fMRI time series," in *Proc. 15th HBM*, San Francisco, CA, USA, juin 2009.

[3] L. Chari, F. Forbes, P. Ciuciu, T. Vincent, and Michel Dojat, "Bayesian variational approximation for the joint detection-estimation of brain activity in fMRI," Tech. Rep., INRIA Rhône-Alpes and LNA0/NeuroSpin, Grenoble, France, fév. 2011.

workflows and pipelines for data processing

Submitted to the Python in Neuroscience satellite to EuroSciPy 2011

Interoperability among Data Processing Frameworks: Reality or Wishful Thinking?

TIZIANO ZITO[a,b], NIKO WILBERT[c,a], RIKE-BENJAMIN SCHUPPNER[a],
ZBIGNIEW JĘDRZEJEWSKI-SZMEK[d], LAURENZ WISKOTT[c,a,e], PIETRO BERKES[f]

[a]*Bernstein Center for Computational Neuroscience, Berlin, Germany*
[b]*Modelling of Cognitive Processes, Berlin Institute of Technology, Germany*
[c]*Institute for Theoretical Biology, Humboldt-University, Berlin, Germany*
[d]*Institute of Experimental Physics, University of Warsaw, Poland*
[e]*Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany*
[f]*National Volen Center for Complex Systems, Brandeis University, Waltham, MA, USA*

May 23, 2011

Experimental and theoretical research in neuroscience constantly requires the application of data processing algorithms for data analysis and modeling. As Python is currently the second-most used programming language in the field, it is relatively easy to find implementations of the most common algorithms.

When building complete applications, scientists are often confronted with several additional chores that need to be carried out beside the individual processing steps. It is common to test different algorithms on the same data set (e.g., in control experiments), or to train and execute a sequence of several algorithms. When the algorithms come from different data processing libraries their implementations likely follow different conventions and do not share a common interface, making their interfacing cumbersome and error-prone.

In this talk, I will discuss the effort taken in the Modular toolkit for Data Processing (**`MDP`**) to ensure the interoperability with other mainstream data processing frameworks. **`MDP`** is a library that provides an implementation of several widespread algorithms, and offers a unified framework to combine them to build more complex data processing architectures. The general strategy to interface with other libraries has been to offer wrappers for the external algorithms. **`MDP`** currently features automatically generated wrappers for **`scikits.learn`**, another widely used data processing framework, and static wrappers for two Support Vector Machine libraries, **`shogun`** and **`libSVM`**. Additionally, **`MDP`** has been designed to be easy to embed in other frameworks, and I will show how this is achieved in **`PyMVPA`**, yet another data processing framework.

Using **`MDP`** as a paradigmatic example, I am going to explore current problems and challenges of achieving interoperability among data processing frameworks and draw some conclusions about the future.

More than batteries included: NeuroDebian

Michael Hanke[1,2,3] and Yaroslav O. Halchenko[1,2]

[1] *Center for Cognitive Neuroscience, Dartmouth College, Hanover, New Hampshire, USA*
[2] *Department of Psychological and Brain Sciences, Dartmouth College, Hanover, New Hampshire, USA*
[3] *Department of Experimental Psychology, University of Magdeburg, Magdeburg, Germany*

NeuroDebian[1] is a turnkey software platform for nearly all aspects of the neuroscientific research process. It helps researchers to cope with the increasing complexity of software systems in neuroscience research. Being embedded in the *Debian*[2] project, it provides a comprehensive suite of readily usable and fully integrated software through a robust deployment infrastructure. Improved system integration and interoperability of research tools frees researchers from the burden of tedious installation or upgrade procedures. That, in turn, positively affects their availability for actual research activities, as well as their motivation to test new analysis tools and stay connected with the latest methodological developments in the field.

Over the past six years, NeuroDebian developers have integrated neuroscience-related software into the Debian operating system – adding to the world's largest archive of integrated free and open source software (FOSS). Debian, with its release record of almost two decades, well-known stability, a unique democratic and open development model, provides an ideal foundation for a reliable and versatile research platform. As a result, the latest release Debian 6.0 "squeeze" offers more out-of-the-box support for neuroscience research than any other operating system[3].

Currently, NeuroDebian offers substantial coverage of MRI-related neuroimaging software, and is expanding its support for psychophysics, electrophysiology and computational neuroscience research. In addition to making software available in the Debian archive, NeuroDebian offers a ready-to-use repository with packages tailored and built for several recent Debian and Ubuntu releases – with repository mirrors in a number of countries. This service is used by hundreds of labs worldwide and serves as the basis of several live-cd and virtual environments, further tailored to various purposes.

This presentation introduces the services offered by the NeuroDebian project, such as backport facilities, package snapshots, communication channels, and virtual machine based neuroscience research environments, that help researchers deal with common day-to-day problems, as well as help improving transparency and reproducibility of research. It showcases how NeuroDebian can facilitate neuroscience software development and deployment – offering benefits for users and developers that reach far beyond the Debian operating system. This includes avoiding licensing problems that hinder adoption of software, identifying and preventing unnecessary dependencies on particular versions of software that make deployment difficult, and continuous integrating testing of whole analysis environments. Complementing the EuroSciPy talk "π's in Debian or Scientific Debian: NumPy, SciPy and beyond"[4] by Yaroslav Halchenko, this presentation provides an overview of the status of Python-based neuroscience software in NeuroDebian, and how it benefits from integration into a larger context of neuroscience research software.

[1] http://neuro.debian.net
[2] http://www.debian.org
[3] http://www.debian.org/releases/stable/i386/release-notes/ch-whats-new.en.html#id401109
[4] http://www.euroscipy.org/talk/4379

NiPyPe: A flexible, lightweight and extensible neuroimaging data processing framework

Over the past twenty years, advances in non-invasive *in vivo* neuroimaging have resulted in an explosion of studies investigating human cognition in health and disease. Current imaging studies acquire multi-modal image data (e.g., structural, diffusion, functional) and combine with non-imaging behavioural data, patient and/or treatment history and demographic and genetic information. Several sophisticated software packages (e.g., AFNI, BrainVoyager, FSL, FreeSurfer, NiPy, R, SPM) are used to process and analyze such extensive data. Current neuroimaging software offer users an incredible opportunity to analyze their data in different ways, with different underlying assumptions. In a typical analysis, algorithms from these packages, each with its own set of parameters, process the raw data. However, data collected for a single study can be diverse (highly multi-dimensional) and large, and algorithms suited for one dataset may not be optimal for another. This complicates analysis methods and makes data exploration and inference challenging, and comparative analysis of new algorithms difficult.

This heterogeneous collection of specialized applications creates several technical, practical and social issues that hinder replicable, efficient and optimal use of neuroimaging analysis approaches: 1) No uniform access to neuroimaging analysis software and usage information; 2) No framework for comparative algorithm development and dissemination; 3) Personnel turnover in laboratories often limit methodological continuity and training new personnel takes time; 4) Neuroimaging software packages do not address computational efficiency; 5) Methods sections in published articles are inadequate for reproducing results; and 6) In-depth knowledge of neuroimaging analysis algorithms is limited to few individuals.

To address these issues, we present NiPyPe (Neuroimaging in Python: Pipelines and Interfaces), an open source, community-developed, Python-based software package that easily interfaces with existing software for efficient analysis of neuroimaging data and rapid comparative development of algorithms. NiPyPe, solves the issues by providing a uniform interface to existing neuroimaging software and by facilitating interaction between these packages using workflows. NiPyPe provides an environment that encourages interactive exploration of algorithms from different packages (e.g., SPM, FSL), eases the design of workflows within and between packages, and reduces the learning curve necessary to use different packages. NiPyPe is addressing limitations of existing pipeline systems and creating a collaborative platform for neuroimaging software development. Processing modules and their inputs and outputs are described in an object-oriented manner providing the flexibility to interface with any type of software. The workflow execution engine has a plug-in architecture and supports both local execution on multicore machines and remote execution on clusters. Python has already been embraced by the neuroscientific community and is rapidly gaining popularity. NiPyPe, based on Python, has immediate access to an extensive community and its software, technological resources and support structure. NiPyPe is distributed with a BSD License allowing anyone to make changes and redistribute it. Development is done openly with collaborators from many different labs, allowing rapid adaptation to the varied needs of the evolving neuroimaging community.

Soma-workflow: An unified and simple interface to parallel computing resources

S. Laguitton[1], D. Rivière[1,2], T. Vincent[1], C. Fischer[1], D. Geffroy[2], N. Souedet[2,3], I. Denghien[2], Y. Cointepas[1,2].
[1]CEA, I2BM, Neurospin, Gif-sur-Yvette, France/[2]IFR 49, Gif-sur-Yvette, France/[3]CEA, I2BM, MIRCEN, Fontenay-aux-Roses, France

Introduction

Neuroimaging research involves many processings applied on high dimensional data and could thus greatly take advantage of the increased availability of parallel computing resources: from multiple core machines to clusters or grids. Coarse-grained parallelism is well suited in most of the cases: it involves many long independent processes (from one minute to several days or weeks). Extensive work to introduce fine grain parallelism in algorithms is indeed worthless when the algorithm is intended to be applied on large sets of data. These coarse-grained parallelized processings can be described by workflows which are sequences of parallel and serial sub-tasks.

Soma-workflow is a Python application which aims at making easier the execution, control and monitoring of tasks and workflows on a wide range of parallel computing resources, through a simple and homogeneous Python application programming interface (API) and/or a graphical user interface (GUI). Soma-workflow was created in the purpose of being plugged to external software or being used directly by non expert users.

Soma-workflow and its interactions with parallel resources

The workflows are described by direct acyclic graphs: the nodes correspond to the sub-tasks and the arrows to the execution dependencies between sub-tasks. Each sub-task, also called job, wraps a program command line call. The workflows are created and then submitted to a computing resource using soma-workflow Python API. For each submitted workflow, soma-workflow deals with the execution of each job at the right time considering its dependencies. The status of workflows and jobs, as well as job standard outputs or exit information, are displayed in the GUI or can be recovered using the API. The workflows can be stopped at any time or/and restarted when needed.

For more flexibility, soma-workflow can be used as a simple single process application or as a client-server application. Compared to the former, the client-server application additionally offers a transparent access to remote computing resources and the possibility for the user to close soma-workflow at any time without impacting the execution of workflows. Tools are also available to handle file transfers and file path matchings in case the user's machine and the remote computing resource do not have a shared file system.

In the purpose of interacting with a wide range of parallel computing resources, soma-workflow uses only very basic parallel resource functionalities: job submission, job suppression and request for job status and exit information. Thereby, a new parallel resource can be plugged to soma-workflow by implementing only four Python methods. Furthermore, the interaction with many computing resources is already implemented in soma-workflow. In the case of a single multiple core machine, soma-workflow can be used directly: the maximum number of jobs which can run in parallel is adjusted by the user. In the case of distributed computing resources, soma-workflow interacts with DRMAA (Distributed Resource Management Application API) [1] which can interface with a wide range of distributed resource management system. Using DRMAA, soma-workflow was used with success on clusters managed by Torque, LSF, Oracle Grid Engine and Condor.

Concrete use cases in neuroimaging

This paragraph illustrates the use of soma-workflow to take advantage of a 192 core shared cluster in three concrete use cases in neuroimaging. In the first use case, soma-workflow was used to execute a coarse-grained parallelized analysis: the joint detection-estimation of within-subject fMRI data [2]. The already existing code was wrapped in a workflow which splits the data, processes it and merges the results afterward. The analysis of a whole brain goes down from 10 hours on a single CPU to 15 mins on the cluster. In the second use case, soma-workflow was used to run regression tests in the process of reducing the sensitivity of the Morphologist pipeline of BrainVISA [3] to variability. The regression test workflow was generated from BrainVISA and consists in the processing of a sample database containing 80 T1 MR images. It runs in about an hour on the cluster against 23 hours on a single processor. In the third use case, soma-workflow was used to perform an extensive cross-validation of the older cortical sulci identification models of BrainVISA [4] . The complete validation represented about 5500 hours of computing (more than 7 months), and used about 70000 individual jobs. It ran in about 3 days using approximately 100 cores of the cluster. The validation brought a precise idea of the performance of the model and allows comparisons with other models such as the methods by Perrot [5].

Conclusion

Soma-workflow aims at filling the gap between parallel computing resources and software or non expert end users. The three neuroimaging use cases presented here demonstrate its relevance in the research process. Indeed, coarse-grained parallelism can exist at the level of the analysis, as in the first use case. However, even if the method cannot be parallelized, the two other use cases show the relevance of using parallel resources to process large datasets, optimize, test and validate methods. Several tracks are currently studied to go further in the flexibility of soma-workflow and usability with a wide range of computing resources.

Acknowledgments. This work was funded by the ITEA2 HiPiP project. Developments relative to the implementation of JDE with soma-workflow were supported by the CL1-38093-007 Servier contract.

References

1. Troger, P., Rajic, H., Haas, A., Domagalski, P.: Standardization of an API for Distributed Resource Management Systems. In: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007). pp. 619–626 (2007).
2. Vincent, T., Risser, L., Ciuciu, P.: Spatially adaptive mixture modeling for analysis of within-subject fMRI time series. IEEE Trans. Med. Imag. 29, 1059–1074 (2010)
3. Cointepas, Y.: The BrainVISA project: a shared software development infrastructure for biomedical imaging research. In: Proceedings 16th HBM (2010)
4. Rivière, D., Mangin, J.-F., Papadopoulos-Orfanos, D., Martinez, J.-M, Frouin, V., Régis, J.: Automatic recognition of cortical sulci of the Human Brain using a congregation of neural networks. In:Medical Image Analysis. vol. 6, no. 2, pp. 77–92 (2002)
5. Perrot, M., Rivière, D., Mangin, J.-F.: Cortical sulci recognition and spatial normalization. In: Medical Image Analysis. In press (2011)

Nitime and IPython: tools for time-series analysis and high-level parallel computing

In this talk we will discuss two different projects, directed at different aspects of data analysis in neuroscience. We will begin with a description of Nitime, a library for the analysis of data from neuroscience experiments. Nitime includes tools for the representation, analysis and visualization of time-series and related quantities. It contains a core of numerical algorithms for time-series analysis both in the time and spectral domains, a set of container objects to represent time-series, and auxiliary objects that expose a high level interface to the numerical machinery and make common analysis tasks easy to express with compact and semantically clear code. The nitime classes can represent time and time-series on several different orders of magnitude (suitable for data from different experimental modalities), as well as events and epochs. In addition, the library contains implementations of algorithms for spectral analysis, event-related analysis and bi-/multi-variate analysis of time-series data. The library contains a high-level, user-friendly API for the scripting of common analysis tasks. This API emphasizes efficiency, in that computations are delayed until they are needed and once they are performed they are cached for further use. Finally, the library contains several visualization functions, based on functionality from Matplotlib and NetworkX.

The talk will focus on the distinction between uni-variate analysis methods, which apply to the characteristics of individual time-series, and multi-/bi-variate methods which apply to the interaction and mutual influence of two or more time-series. In order to demonstrate this distinction, we will present multi-variate analysis of data from functional Magnetic Resonance Imaging (fMRI) experiments. We will contrast and compare three methods for the calculation of functional connectivity between voxels in fMRI experiments: correlation, spectral coherency and multi-variate autoregressive analysis (“Granger causality”).

The second part of the presentation will focus on IPython’s new API for high-level parallel computing. As of this year, IPython has a completely new backend to provide easy-to-use distributed and parallel computing tools, based on the high-performance ZeroMQ networking library. With the rapid rise of multicore systems and clusters, the flattening of the speed curve for modern microprocessors and the ever-increasing sizes of datasets in neuroscience, most scientists today will need to parallelize many of their analyses in everyday work. Yet, classical parallel computing tools tend to be cumbersome to use, deploy and interact with, as they have been classically tuned for absolute performance at all costs, at the detriment of usability. IPython, instead, tries to provide a very high-level, easy to understand and use model for the parallelization of common tasks, retaining enough performance to be useful in production contexts but with a constant concern for the scientist’s productivity rather than absolute computational performance. We will present a description of the main concepts involved, the abstraction model offered by IPython and some simple examples of practical usage.

Nitime can be found at: http://nipy.org/nitime and IPython at: http://ipython.org.

“visualization tools”

Anatomist: a python framework for interactive 3D visualization of neuroimaging data

D Rivière[1,2], D Geffroy[1], I Denghien[1], N Souedet[1,3], Y Cointepas[1,2]

[1]IFR 49, Gif-sur-Yvette, France/[2]CEA, I2BM, Neurospin, Gif-sur-Yvette, France/[3]CEA, I2BM, MIRCEN, Fontenay-aux-Roses, France

Introduction

Anatomist is a powerful 3D visualization software dedicated to neuroimaging. It is cross-platform, open-source, distributed under the CeCill-B license [1] (similar to BSD), and available on BrainVISA [2] web site (http://brainvisa.info). Originally developed in C++ language for about 15 years, a set of python bindings and extensions have then been developed for Anatomist, including integration with popular python modules like Numpy [3], Matplotlib [4], or IPython [5].
Anatomist technologies rely on OpenGL [6], Qt [7] and PyQt [8] for the graphics and 3D parts, and on AIMS / PyAims libraries for neuroimaging data structures, IO and algorithms. It is able to display an arbitrary number of objects in an arbitrary number of views which may be embedded in dedicated graphical interfaces. The AIMS library (also part of the BrainVISA project) offers many plugin-based IO formats for several data structures: 3D/4D volumes (supporting DICOM, NIFTI, MINC, ECAT, and many 2D image formats), surfacic meshes (supporting GIFTI, PLY, MNI-Obj and a few other export formats like VRML-1 and POV), textures, voxels lists, more complex structured container objects (graphs, used for ROI drawing, sulci and fibers representation, or cortex parcellations), and a subset of VTK [9] objects. Objects may be combined to form new objects mixing properties of the combined ones, which offer richer display modes, such as overlaying volumes, texturing meshes according to 3D or 4D volumic data and/or to raw texture data, cutting meshes with volumic data along an arbitrary plane, volume rendering, clipping, and so on. Such functionalities may be extended by plugins.

Python control modes

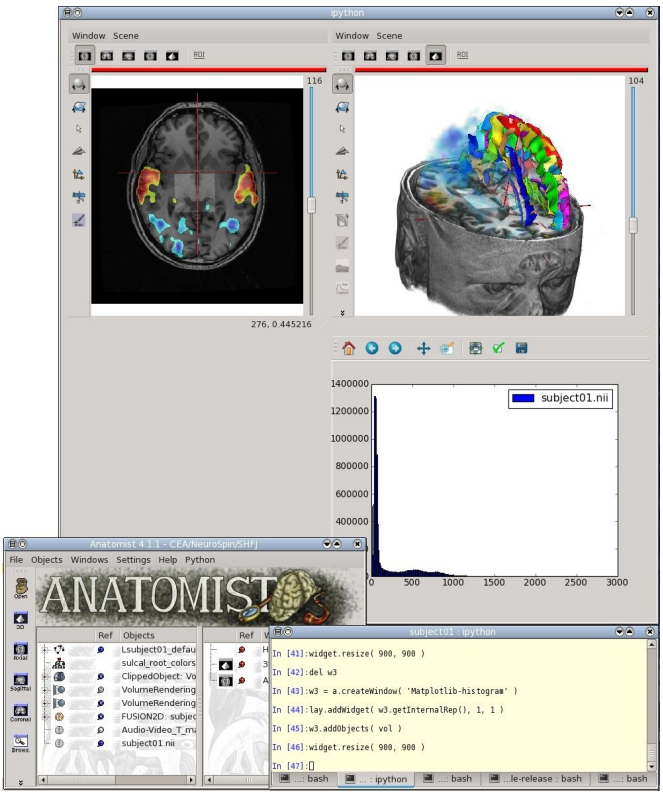
The python API of Anatomist allows two main modes: one is the “direct bindings” mode which offers an access to the underlying C++ data structures in memory, and the other one, “socket” allows to control a remote Anatomist as a separate application via a network connection. Both modes are using the same API, except that the “direct” mode is richer and allows more functionalities, such as GUI embedding and direct data modification in memory for visualization update on the fly.
The high-level python API is based on an “application” object which may address a unique singleton in direct mode application, and possibly several socket-based applications. Through this application object, manipulations use simple functions like `loadObject()`, `createWindow()` and so on (Fig. 1). Objects and windows are handled through dedicated classes (`AObject` and `AWindow`, namely). These objects grant access to various properties: colormaps, materials, referentials, camera properties, display modes, and other parameters.
Documentation can be found at http://brainvisa.info/doc/pyanatomist/sphinx/.

Lower-level API for full customization

In addition to this high-level API, intended to be quickly usable even by novice programmers, the “direct” mode offers an additional lower-level API which is actually the direct bindings of C++ classes. More complex due to its richness, this level of programming offers the possibility to extend many of Anatomist mechanisms: new object types and new window types, may be defined by subclassing in Python the C++ classes. 3D graphical objects may be defined at various levels, either reusing parts of existing graphical primitives of existing objects (geometric properties, vertices etc), or completely defining the 3D rendering at OpenGL level. Interaction controls (keyboard and mouse actions for users) and callbacks may be defined. Complete dedicated GUI applications may be designed. For this last usage, it is possible to use Qt Designer [7] to graphically design the application user interface. The anatomist main window can be suppressed to be replaced by a custom GUI, or by only an interactive python shell if needed.

Conclusion

Anatomist offers a wide range of possibilities for controlling, designing and extending 3D visualization for neuroimaging data in Python language. Combining the existing visualization and interaction features already provide numerous and very powerful capabilities at a low programming cost. Different complexities of customization are addressed at different levels of API to keep a high-level layer as simple as possible, thus still allowing to extend mechanisms at lower level and at lower scales, possibly down to the OpenGL layer.




```python
import anatomist.direct.api as ana
import numpy
from PyQt4 import QtGui

# main application
a = ana.Anatomist()
# load an anatomical MRI volume
vol = a.loadObject( 'subject01.nii' )
# setup normalization transformation and referentials
vol.applyBuiltinReferential()
# load a functional T map
func = a.loadObject( 'Audio-Video_T_map.nii' )
func.applyBuiltinReferential()
# display both with appropriate colormaps
f = a.fusionObjects( (vol, func), method='Fusion2DMethod' )
func.setPalette( 'tvalues100-200-100-lfusion', minVal=-5, maxVal=5,
  absoluteMode=True )
widget = QtGui.QWidget() # we use a custom Qt widget to group several views
lay = QtGui.QGridLayout( widget )
w1 = a.createWindow( 'Axial' ) # anatomist window
lay.addWidget( w1.getInternalRep(), 0, 0 )
widget.show()
w1.addObjects( f ) # display fusionned object (anat + functional volumes)
# go to Talairach position (0,0,0)
a.execute( 'LinkedCursor', window=w1, position=(0, 0, 0) )
# use some volume rendering
vr = a.fusionObjects( vol, method='VolumeRenderingFusionMethod' )
vrf = a.fusionObjects( func, method='VolumeRenderingFusionMethod' )
# with clipping on the anatomical volume
cvr = a.fusionObjects( vr, method='FusionClipMethod' )
# display both
w2 = a.createWindow( '3D' )
lay.addWidget( w2.getInternalRep(), 0, 1 )
w2.addObjects( ( cvr, vrf ) )
# load a sulci nomenclature and graph
nom = a.loadObject( 'sulci/sulcal_root_colors.hie' )
sulci = a.loadObject( 'sulci/Lsubject01_default_session_auto.arg' )
sulci.graph()[ 'label_property' ] = 'label'
sulci.setReferential( vol.referential )
sulci.notifyObservers()
w2.addObjects( sulci, add_graph_nodes=True )
# rotate the view for better visibility
w2.camera( view_quaternion=[0.484051, -0.196338, -0.156624, 0.838222] )
# use matplotlib-based histogram module
w3 = a.createWindow( 'Matplotlib-histogram' )
lay.addWidget( w3.getInternalRep(), 1, 1 )
widget.resize( 900, 900 )
w3.addObjects( vol )
# now modify the anatomical volume data using numpy
numpy.array( a.toAimsObject( vol ).volume(), \
  copy=False )[100:150,100:150,40:80,0] += 300
vol.setChanged()
vol.notifyObservers()
```

Fig. 1: Example of Anatomist handling in python, showing a script (right) which may be used in an interactive IPython shell (left, lower), and the visual result (left). Views may be interactively modified, rotated, zoomed etc.

References:

[1] CeCILL licences, http://www.cecill.info/index.en.html
[2] BrainVISA, http://brainvisa.info
[3] Numpy, http://numpy.scipy.org
[4] Matplotlib, http://matplotlib.sourceforge.net
[5] IPython, http://ipython.scipy.org
[6] OpenGL, http://www.opengl.org
[7] Qt, http://qt.nokia.com
[8] PyQt, http://www.riverbankcomputing.co.uk
[9] VTK, http://www.vtk.org

“Poster”

Waxholm space

Raphaël Ritz, INCF Secretariat

International Neuroinformatics Coordinating Facility
Date: 2011-01-10
Reference number of this INCF document: INCF 10-001
Version: 0.7
Category: INCF Discussion Paper
Editor: Ilya Zaslavsky

INCF Atlas Services and Waxholm Markup Language

Digital Atlasing Infrastructure Task Force

3D Brain Atlas Reconstructor and Common Atlas Format, the infrastructure for constructing three dimensional brain atlases

Piotr Majka, Ewa Kublik, Jakub Kowalski, Daniel K. Wójcik
Department of Neurophysiology, Nencki Institute of Experimental Biology, 3 Pasteur Street, 02-093 Warsaw, Poland

Anatomical brain atlases are essential part of neuroscientific toolbox. There is a growing demand for extending typical two dimentional atlas frameworks into the three-dimensional space. In response to such demands we propose a Python package for automated reconstruction of 3D models of brain structures basing on their 2D delineations of varying formats, sophistication and quality.

The 3d Brain Atlas Reconstructor (3dBAR, http://www.3dbar.org) contains a set of generic Python parsers for converting source atlases into Common Atlas Format (CAF), Python API for manipulating CAF datasets, and Python reconstruction module for generating 3D models. All segments of software were prepared in object-oriented manner to simplify code maintenance and extensibility. 3dBAR relies fully on open file formats (XML, SVG, VRML, NIfTi) and open source packages and will also be open. 3D graphics processing is performed using Visualization ToolKit (http://www.vtk.org/). SVG rasterization and image manipulation is handled by python-rsvg library (http://cairographics.org/pyrsvg/), Python Image Library (http://www.pythonware.com/products/pil/) and SciPy. Graphical User Interface was prepared using wxPython (http://www.wxpython.org/). NifTi export is based on PyNIfTI (http://niftilib.sourceforge.net/pynifti/).

Reconstruction process leading to a three dimensional model proceeds in two steps. First the source atlas is parsed into a Common Atlas Format dataset. Every CAF dataset consists of SVG graphics files and an XML index file. SVG files contain 2D representations of brain slices divided into separate structures represented by closed paths and an information about the brain's spatial coordinate system. Structures' hierarchy and additional informations (e.g. structures full names, external references to databases) are stored in the index file. The second stage involves processing CAF dataset and results in a 3d reconstruction in the form of a volumetric data or polygonal mesh and it is realized with an user friendly GUI written in Python.

The 3dBAR uses automatic, customizable and reproducible workflow which allows tracking and reviewing of the whole reconstruction process as well as locating and eliminating reconstruction errors or data inconsistencies. Thank to it's open format policy it supports interoperability with other neuroinformatics tools and exchange of content at any stage of processing.

We have used this infrastructure to process several widely used brain atlases, including published atlases with PDF datasets (“The Rat Brain in Stereotaxic Coordinates” by Paxinos and Watson and “The Mouse Brain in Stereotaxic Coordinates” by Paxinos and Franklin) or prepared from volumetric data (The Waxholm Space Atlas; Johnson et al, 2010). Five other datasets were derived from the Scalable Brain Atlas (SBA, http://scalablebrainatlas.incf.org).

Apart from standalone package we provide a web service (http://www.3dbar.org) that allows a variety of operations on available CAF datasets, e.g. customizable reconstructions, downloads in various formats, viewing thumbnails of structures, customized viewing of slides from provided CAF datasets and access to structure hierarchy. Live view of 3D structures is also possible in WebGL enabled browsers where one can manipulate (rotate, zoom, etc.) lightweight versions of the reconstructions.

Gom2n: A toolchain to simulate and investigate selective stimulation strategies for FES

Jeremy Laforet

When contracting a muscle using NFES (Neural Functional Electrical Stimulation), the stimulus always activates the axons of greater diameter first. Also selective activation of given fascicle inside a nerve is not possible with classical cuff electrode as the recruitment is performed uniformly around the nerve. These limits lead to poorly selective muscle recruitment, inducing fatigue and possible pain. To overcome this, selective stimulation strategies can be used.

We already developed multipolar electrodes and benchtop stimulators with industrial partners to provide for future implantable stimulation units, with up to 12 poles, and arbitrary stimulation shape [1]. Even though literature describes the principles of the strategies to be used, tuning remains a tricky problem. This is the reason why we develop a modelling toolchain, not only to simulate and investigate selective stimulation, but also to optimise it, mainly as regards charge injection minimisation, and selectivity level.

We propose a toolchain [2] to investigate, simulate and tune selective stimulation strategies. It relies on OpenMEEG [3] and Neuron [4] and consists of a conduction volume model to compute the electric field generated in the nerve by a cuff electrode surrounding it; an axon model to predict the effect of the field on the nerve fibre — the generation, propagation and possible block of action potentials.

Gom2n includes a graphical interface to enable to use of the toolchain to a wider audience. It is developped in Python, using PyGtk and matplotlib. All parameters can be visualised and set in the interface. It is also possible to resume previous simulations or to reuse part or all of the model descriptions.

Many improvements are in progress : optimisation of the injected currents vector for spatial selectivity, tools to assist setting the waveform for anodal block, and batch simulations. Some work has been done also to adapt the toolchain for cochlear implants tuning [5].

References

[1] David Andreu, David Guiraud, and Guillaume Souquet. A distributed architecture for activating the peripheral nervous system. *J Neural Eng*, 6(2):26001, Apr 2009.

[2] Jeremy Laforet, David Guiraud, and Maureen Clerc. A toolchain to simulate and investigate selective stimulation strategies for fes. In *Conf Proc IEEE Eng Med Biol Soc. 2009*, pages 4966–9, 2009.

[3] Alexandre Gramfort, Theodore Papadopoulo, Emmanuel Olivi, and Maureen Clerc. Openmeeg: opensource software for quasistatic bioelectromagnetics. *BioMedical Engineering OnLine*, 9(1):45, 2010.

[4] M. L. Hines and N. T. Carnevale. Neuron: a tool for neuroscientists. *Neuroscientist*, 7(2):123–135, Apr 2001.

[5] Jeremy Laforet, Jessica Falcone, Nicolas Veau, and David Guiraud. Gom2n : a software to simulate multipolar neural stimulation for cochlear implants. In *IEEE EMBS Conference on Neural Engineering*, 2011.

Developing and evaluating a computerized tool for measuring perceived stress

Dalal Ben Loubir, Medical Informatics Laboratory, Faculty of Medicine and Pharmacy of Casablanca, Morocco.
Professor Mohammed Bennani Othmani, Medical Informatics Laboratory, Faculty of Medicine and Pharmacy of Casablanca, Morocco.

Stress is a component among others of depressive illness that take place more and more in our society(family, work, school…), in addition, it is the subject of several research for decades. Stress is related to coping abilities; it may bring changes on individual's physical and mental well-being.

This work is a part of a thesis, which will end in 2013. Our main aim is to create a homogenization between computing and neurological fields. From this work we try to produce an evaluative ICT tool that could be incorporated as part of clinical practice.

The main goal of this study is to develop and evaluate an application that will support the measure of perceived stress in individuals.

Currently, we are working on a project which covers the study of stress and performance in order to better assess stress response in medical students who face the charge of the program and exam preparation.

In a second step, we will explore the concept of perceived stress to identify the dimensions of the concept.

In the last step, we will develop a computer application for measuring the perception of stress in individuals.

Keywords: Stress, Stress Perception, Computerized Tool

Simulating topographic distributions of event-related potentials using Brisk

Roman Goj, David Donaldson, Mark van Rossum

June 8, 2011

We present an early version of an event-related potential (ERP) simulation tool written entirely in Python. Brisk (BRain Imaging Simulation Kit) currently enables users to simulate the topographic distribution of ERP activity over the scalp as well as control several aspects of between-subject variability in the activity of the underlying neuronal generators. Our aim is to develop a complete ERP phantom for use in research and education.

Electroencephalographic (EEG) recordings, especially when averaged across multiple experimental trials and multiple participants to reveal ERP signals, enable quantification of the relation between human cognition and brain activity. Analyses of ERP data revolve around complex statistical comparisons of ERP component properties, such as amplitude or topographic shape, between experimental conditions. Although simulation-based approaches are often adopted for justification and verification of ERP data analysis methods, the assumptions underlying the simulations are typically either identical to those inherent in a given data analysis method or simply not stated clearly enough to allow reproducibility. We aim to address both of these issues by providing a software tool which can be used as an ERP phantom – enabling reliable verification of data analysis methodology.

Our current focus has been on between-subject variability in topographic patterns of ERP activity. Most ERP studies rely on data collected from and averaged across multiple participants. However relatively little is known about the between-subject differences in neuronal generators, and hence topographic distributions of ERP activity. Typically simulations assume identical neuronal generator configurations across all subjects. All between-subject differences are assumed to be accounted for by normal noise with equal variance across all subjects and electrodes. Visual inspection of topographic distribution plots reveals striking differences between data simulated using the above described simulation methodology and ERP recordings collected in real experiments. We provide alternative approaches to ERP simulation, implemented within the Brisk software, that rely on variability in generator amplitude to better reproduce properties of topographic distributions of real ERP signals.

Although accurate simulation of EEG and ERP activity may yet require many years of research, we believe that in order to facilitate this research a working simulator, clearly stated assumptions, and their clean and easy to modify implementation are necessary. The presented early version of the Brisk simulator is a first step in this direction. Our long term goal is to include an intuitive graphical user interface that would enable both students and experienced ERP researchers to visually explore the parameters of the ERP simulation process, building intuitions about the brain and its relation to cognition along the way.

A PYTHON Package for Kernel Smoothing via Diffusion: Estimation of Firing Rate from Spike Trains

Taskin Deniz *, Stefano Cardanobile, Stefan Rotter
Berstein Center Freiburg & Faculty of Biology, University of Freiburg, Germany

Kernel smoothing is a powerful methodology to gain insight into data. It has wide applications in many different fields, ranging from Economics to Neurosciences. The most important basic application of kernel smoothing in Neuroscience is estimation of time-dependent firing rates from neuronal spike trains. Traditionally, this is achieved by the PSTH (Peri-Stimulus Time Histogram) or, alternatively, smoothing with a fixed kernel. The PSTH relies on the availability of multiple trials for averaging out trial-to-trial fluctuations. However, one can obtain a plausible estimate from a single trial as well, using kernel smoothing methods, where the bandwidth of the kernel is a parameter to be selected in analogy to the bin size of the histogram. The form of the kernel is rather unimportant, provided it is smooth, unimodal and normalized. Its bandwidth, in contrast, defines how smooth the resultant rate would be (Nawrot et al., 1999). A suboptimal kernel may result in over-smoothing or under-smoothing, where the optimal kernel is defined by a minimal deviation from the true rate profile. There may be no globally optimal kernel for strongly changing Poisson rates, though. As a cure to this problem one can optimize the estimate by locally adaptive bandwidth selection. To this end, Shimazaki and Shinomoto (2009) suggested a combinatorial way of optimizing MISE (mean square integrated error) as a method of local bandwidth estimation. This method, although effective, is computationally very costly and biased. Instead, we suggest an application of a new method by Botev et al. (2010), namely Kernel Density Estimation via Diffusion. The diffusion method offers a fast completely data driven algorithm for local bandwidth selection, avoiding the boundary bias and the assumption of Gaussianity. An implementation of the new method as a PYTHON package is made available.

References

1. Nawrot M, Aertsen A, Rotter S (1999) Single-trial estimation of neuronal firing rates - From single neuron spike trains to population activity. *Journal of Neuroscience Methods* 94(1): 81-92
2. Botev ZI, Grotowski JF, Kroese DP (2010) Kernel density estimation via diffusion. *Annals of Statistics* 38(5): 2916-2957
3. Shimazaki H, Shinomoto S (2009) Kernel bandwidth optimization in spike rate estimation. *Journal of Computational Neuroscience* 29(1-2): 171-182
4. Jones MC, Marron JS, Sheather SJ (1996) A Brief Survey of Bandwidth Selection for Density Estimation. *Journal of the American Statistical Association* 91(433): 401-407

Funding by the German Ministry of Education and Research (Bernstein Focus Neurotechnology Freiburg*Tübingen, FKZ 01 GQ 0830) is gratefully acknowledged.

* Corresponding Author, email: taskin.deniz@bcf.uni-freiburg.de

Random Subspace Methods for Neuroimaging

Diego Sona[1,2] and Paolo Avesani[1,2]
[1] *Neuroinformatics Laboratory, Fondazione Bruno Kessler, Italy*
[2] *Center for Mind/Brain Sciences, University of Trento, Italy*
{*sona,avesani*}*@fbk.eu*

Functional Magnetic Resonance Imaging (*fMRI*) allows for a non-invasive indirect measurement of the brain activity with a relatively high spatial resolution. This technology locally records the variation in time of the oxygenated blood flow, called *Blood Oxygenation Level Dependent* (*BOLD*), which is somehow related to the variation of the local brain activity. The measurement is taken as a sequence of volumetric brain images, usually few hundreds, each made of several thousands voxels. The voxels are the smaller portion of brain (usually 2-3 millimeters diameter) where *BOLD* signal can be recorded.

The typical experiment consists in a stimulation protocol with contrasts that maximally activate some cognitive functions. The recorded data is then analyzed to locate the portion of brain related to the specific cognitive task. This analysis, known as *brain mapping*, produces a volumetric image where the voxels are coloured according to their relevance.

In a pattern recognition framework, brain mapping can be conceived as a problem of feature selection. There are basically three approaches to feature selection: *embedded*, *wrapper*, and *filter* methods. Differently from many other domains, the selection of features in the brain mapping framework can not be driven by the performance of a classifier, like in *embedded* and *wrapper* methods. The main requirement here is the redundancy preservation, i.e., all relevant voxels must be retrieved, even if redundant. For this reason, the most appropriate approaches appears to be the *filter* methods, where the features are ranked according to some information-based scores or statistical indices. As a matter of fact, the most common brain mapping approach consists in a uni-voxel linear analysis (commonly referred as *GLM* analysis) with a statistical test over it. The *filters* approach has, however, some limitations: cross-relationships between different parts of the brain may remain undiscovered, moreover, the retrieved features are good descriptors not necessarily allowing for good discrimination between conditions.

The increasing interest on distributed patterns of activity is rising the attention of the neuroscience community to the *Multi-Voxel Pattern Analysis* (*MVPA*) methods, where many voxels are jointly analyzed with discriminative methods. The typical approach aims at predicting the cognitive states with a classifier given the brain activity [1]. This approach, however, does not directly answer the brain mapping issue, which is usually addressed ex post, on the trained classifiers, determining the relevance of the used voxels for the classifier itself.

The main challenge of this approach is to deal with the huge dimensionality of the features space, and the excessive feature-to-instance ratio. There are few *MVPA* solutions directly addressing this task. The *Elastic-Net* [2] and the *Recursive Feature Elimination* [3] perform a whole-brain analysis, finding distributed relationships. These *embedded* approaches, however, suffer of stability problems (i.e., two different runs may produce two completely different solutions). The well known and robust *Searchlight* method [4], on the other side, perfectly copes with the stability issue finding local patterns of activity. The method, however, still suffer some limitations: the spatial bias prevents the detection of distributed pattern of activity; the brain maps overestimate the areas of activation; there is no distinction between correlated and anti-correlated areas; and a whole-brain analysis is not supported.

To cope with all above problems we proposed a wrapper-based solution exploiting a *Random Subset Method* (*RSM*) for feature selection [5]. This multi-variate method is essentially an ensemble method exploiting a bootstrapping principle. Many classifiers are trained with different random subsets of the feature space. It may be considered a *wrapper* method generalizing *Searchlight* to a distributed analysis. The proposed solution presents therefore some advantages. It performs a multi-variate analysis of the whole brain dealing with the unfavorable ratio between voxels and volumes, and it preserves the voxels redundancy. We have proven that under some conditions the method provides results equivalent to the standard *GLM* analysis.

The model has been implemented using python, and exploiting the pyMVPA v 0.4.7 library, both for the management of the neuroimaging data and for the exploitation of the classification framework. The library providing the random subspace method will be deployed as open source software.

References

[1] K. Norman, S. Polyn, G. Detre, and J. Haxby, "Beyond mind-reading: multi-voxel pattern analysis of fmri data," *Trends in Cognitive Sciences*, vol. 10, no. 9, pp. 424–430, 2006.

[2] M. K. Carroll, G. A. Cecchi, I. Rish, R. Garg, and A. R. Rao, "Prediction and interpretation of distributed neural activity with sparse models." *NeuroImage*, vol. 44, no. 1, pp. 112–22, 2009.

[3] F. De Martino, G. Valente, N. Staeren, J. Ashburner, R. Goebel, and E. Formisano, "Combining multivariate voxel selection and support vector machines for mapping and classification of fMRI spatial patterns." *NeuroImage*, vol. 43, no. 1, pp. 44–58, 2008.

[4] N. Kriegeskorte, R. Goebel, and P. Bandettini, "Information-based functional brain mapping." *Proc. of the National Academy of Sciences of the United States of America*, vol. 103, no. 10, pp. 3863–3868, 2006.

[5] D. Sona and P. Avesani, "Feature rating by random subspaces for functional brain mapping," in *Int. Conf. on Brain Informatics (BI)*, ser. LNCS, vol. 6334. Springer, 2010, pp. 112–123.

Machine learning for fMRI in Python: inverse inference with scikit-learn

The scikit-learn (Pedregosa 2010) is a general-purpose package for simple and efficient machine learning in Python. It can easily be combined with the various powerful Python packages for fMRI data processing, such as nipy (Brett 2007) or nibabel (Hanke 2009), to apply state-of-the-art learning methods to inverse inference, i.e. the prediction of behavior from brain imaging. The variety of supervised and unsupervised methods available in the scikit-learn as well as their performance greatly facilitates the use of machine learning for brain mapping. In addition, algorithms in the scikit-learn can be used as building-blocks to develop fMRI-specific statistical learning methods.

In the supervised learning settings, of particular importance for the brain-imaging community are the SVM-based methods and the penalized GLMs. For SVMs, the scikit-learn features libSVM bindings -amongst the richest and most performant- as well as associated methods such as recursive feature elimination. Penalized-GLM methods are available, to inject multivariate priors in a GLM, for regression or for classification (logistic regression); in particular L1 penalization and ARD priors are available to learn sparse predictive spatial maps. The scikit-learn also offers tools for cross validation and model selection to automatically set parameters or compare methods. For this, it can make use of multiple processors in a same computer. Unsupervised methods include clustering algorithms, mixture modeling as well as PCA and ICA.

Special care is taken across the scikit-learn code to present the various methods with a uniform interface, in order to make them easily interchangeable and to facilitate comparisons.

Finally, a documentation is available on the web site with a large number of examples and a special attention to ease of read for non machine-learning specialists. The scikit-learn heavily relies on the numpy package, which makes it possible to use the Python language as an array-based scientific computing environment, similar to matlab. Thus the scikit-learn can easily be picked up by matlab users who do not know Python.

We have implemented various fMRI-specific methods using the scikit-learn (Michel 2011a, Michel 2011b) and have used the scikit-learn to compare them to standard state-of-the-art learning methods. A full second-level inverse-inference pipeline, starting from first-level beta maps saved as Nifti file and extracting predictive brain features as well as cross-validated prediction scores can be written in 50 lines of code (Gramfort 2010). In addition, we use unsupervised learning methods to model resting-state functional connectivity, using ICA (Varoquaux 2010a) or covariance estimation (Varoquaux 2010b).

By bridging the gap between state-of-the-art machine-learning research and its applications to brain imaging, the scikit-learn can be used for multivariate brain mapping with high numerical and prediction performance, but also to develop new inverse-inference methods specially crafted to answer the needs and features of neuroimaging.

References

- Brett, M. (2007), 'nipy: NeuroImaging in Python', http://nipy.sourceforge.net/
- Gramfort, A. (2010), 'NeuroImaging with the Scikit-learn: fMRI inverse inference tutorial', http://nisl.github.com/
- Hanke, M. (2009), 'nibabel: Access a cacophony of neuro-imaging file formats', http://nipy.sourceforge.net/nibabel/
- Michel, V. (2011a), 'Total variation regularization for fMRI-based prediction of behaviour', Trans. in Med. Im. in press
- Michel, V. (2011b), 'A supervised clustering approach for fMRI-based inference of brain states', Pattern recognition, in press
- Pedregosa, F. (2010), 'scikit-learn: Machine Learning in Python', http://scikit-learn.sourceforge.net/

- Varoquaux, G. (2010a), 'A group model for stable multi-subject ICA on fMRI datasets', Neuroimage 51, pp288-299
- Varoquaux, G. (2010b), 'Detection of brain functional-connectivity difference in post-stroke patients using group-level covariance modeling', MICCAI

Dynamical characterisation of neural networks and neurophysiological time series: Parallel approaches using Python

Thomas Greg Corcoran

08 June 2011

t.corcoran@susx.ac.uk
Centre for computational neuroscience and robotics (CCNR),
University of Sussex, Brighton UK

Abstract

This work is an exploration of the use of time-delay embedding to explore the dynamical properties of neural network simulations and physiological signals. Whilst parallel approaches such as MPI and CUDA are extremely useful in the simulation of neural systems, the construction of statistics from such simulations can be even more computationally challenging.

Reconstructed attractors (via Takens embedding [1]) are naturally immutable data structures, thus making their analysis an excellent candidate for parallel and/or functional approaches. Moreover, the calculation of statistics of attractors, such as finite-time Lyapunov exponents (FTLE) and entropies [2], are typically very time-consuming. Therefore a toolbox of parallel algorithms is desirable.

This work compares and explores the use of python multiprocessing (chunk-based parallism) and Thrust/PyCuda [3,4] (fine-grained parallelism via GPGPU) approaches to the problem.

The outcome of this work is a Python package for the dynamical characterisation of neuro-physiological and simulation timeseries, bridging the dynamical and statistical properties of neural systems. These indices are useful in the multiscale characterisation of activity patterns, dimensional reduction of models, and distinguishing of chaotic from stochastic systems.

References

1. Takens F. Detecting strange attractors in turbulence. Dynamical systems and turbulence. 1981

2. Castiglione P, Falcioni M, Lesne A, Vulpiani A. Chaos and Coarse Graining in Statistical Mechanics. Cambridge University Press; 2008.

3. Jared Hoberock and Nathan Bell. Thrust: A Parallel Template Library. http://www.meganewtons.com/. 2010

4. Pinto N, Lee Y, Catanzaro B, Ivanov P, Fasih A. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. arXiv. 2009

Authors Index