Python in neuroscience

Martin Spacek and Nicholas Swindale

A widely used open-source programming language, Python is becoming the language of choice for neural data analysis and simulation.

Ideally, neural analysis and simulation code should be simple to write and execute, yet should also be understandable months and years later when it needs to be rerun or modified. A scripting language with an interactive command-line environment, such as MatLab (The MathWorks, Natick, MA, USA), is well suited to quickly implementing ideas. However, MatLab is an expensive closed-source commercial product. This hinders sharing of scientific code. Increasingly, the field has been turning to the Python programming language, which offers all of the benefits of MatLab plus many more, in a free open-source environment.

Similar to MatLab’s toolboxes, Python has many scientific packages that together provide a wide array of capabilities, from algebra to Z-transforms. Some already exist specifically for neural data analysis and simulation, such as PyMVPA and Brian respectively. However, unlike MatLab, Python is also used extensively outside of scientific circles in more general applications such as web development and database management. This gives it a much bigger user base, and therefore much greater scrutiny. Python’s syntax is simple, consistent, and reads like formalized pseudocode. Yet the language is fast and powerful, combining many of the best features from other languages, and allowing integration of foreign code and libraries.

Perhaps most significantly, Python is a fully object-oriented language. Scientific code is often written in a throwaway manner to quickly obtain results. But as analyses and simulations become more complex, code reuse becomes increasingly important. Object-oriented programming encourages code reuse by taking advantage of our natural ability to conceptualize physical objects with different properties and behaviours, and to group those objects into a hierarchy of classes. An object is an instance of a class, and in Python an object’s properties and behaviours are called attributes and methods respectively. Everything in Python is an object, even simple strings and numbers, and therefore everything has attributes and methods.

We needed a way to perform a variety of custom analyses on our data, which consist of multiple-unit extracellular silicon polytrode recordings in cat primary visual cortex, with responses elicited by visual stimulation, and raw waveforms sorted into multiple unit spike trains. The Python package that resulted, Neuropy, is centered around a hierarchy of data classes suitable for our datasets (Figure 1). At the top is an abstract Data class that can access all of the data. This is partitioned into data from different Animals, from which Recordings may have been made at various polytrode positions (Tracks). Finally, each Recording can have multiple spike Sorts performed on it, yielding a list of Neurons each with its own list of spike times. A Recording may also have a visual stimulus Experiment that was run concurrently. Sorted spike data and stimulus information are retrieved from disk from files and folders with a stereotyped naming scheme. Elements in the hierarchy can easily be

![Figure 1. Neuropy’s data class hierarchy. An instance of each class can contain multiple instances of classes below it. Adapted from previous work.](image)

Continued on next page
modified or rearranged to better correspond to different datasets, such as those generated from neural simulations or chips.

A data-centric organization makes analyses easier to write and execute. Analyses are accessible as methods of one or more classes in the hierarchy. For example, running the spike-triggered average (STA) analysis method of a stimulus Experiment object causes each of the child Neurons of the Experiment’s parent Recording to run its own STA method, the results of which are all collected in a single list. Often, analyses themselves are written as their own classes. When requested, an analysis object of that class is returned, and its various input parameters (stored as attributes) can be inspected and modified as needed before running its calc() method. Results are saved to one or more attributes of the analysis object, and can be inspected or plotted. Multiple plots can be generated using different plot parameters without having to re-run the analysis. This encourages experimentation at the interactive command line, without unnecessary computational delays. Currently implemented analyses include interspike interval histograms, instantaneous firing rates and their distributions, cross-correlograms, autocorrelograms, STAs, and several analyses related to population network states. Stimulus tuning curves and peristimulus time histograms will be easy to add when needed.

Figure 2 shows example interactive Python code that runs Neuropy’s STA analysis. First, all of its functionality is made available in the local namespace (line 1). Next, an instance of the Recording class is created and assigned a name (r92 in this case). An ID is passed as an argument to the class to specify which Recording to create an instance of. This implicitly requests a recording from the default Animal and Track, as well as the default Sort beneath that recording (see Fig 1). Next, all of the actual spike and stimulus data are loaded from disk by calling the recording’s load() method (line 3). Inspection of the recording’s e attribute (lines 4–5) reveals that only one stimulus Experiment, with ID 0, is associated with this recording. Further inspection would reveal that this is a white-noise m-sequence experiment, and is therefore suitable for computing the STA. Experiment 0’s sta method is called on line 6, requesting the STAs for neurons 2 and 5. The result is returned as an STAs analysis object (line 7) that has a stas attribute, which itself contains a list of the STA analyses of each individual neuron (lines 8–10). Finally, calling the STAs object’s plot() method displays a window showing the spatiotemporal STAs of both neurons over default time ranges following stimulus presentation. The analysis naming conventions used here are fairly arbitrary, and can be modified to whatever is deemed most intuitive.

Changing programming environments can be a major decision, but it need not be an all-or-nothing process. We feel that coding in Python is more enjoyable, and that it takes significantly less time and effort to write and maintain Python code, resulting in greater productivity. Switching from MatLab has allowed us to tackle more complicated software projects with relative ease. The field of neuroscience is moving towards greater adoption of Python, which should help encourage collaboration through greater sharing of code.

This work was supported by Canadian granting agencies CIHR and NSERC.

Author Information

Martin Spacek and Nicholas Swindale
Ophthalmology and Visual Sciences
University of British Columbia
Vancouver, BC, Canada
http://swindale.ecc.ubc.ca

Martin Spacek is a PhD candidate in neuroscience.

Nicholas Swindale is a professor of Ophthalmology and Visual Sciences.

Continued on next page
References


