SPySort

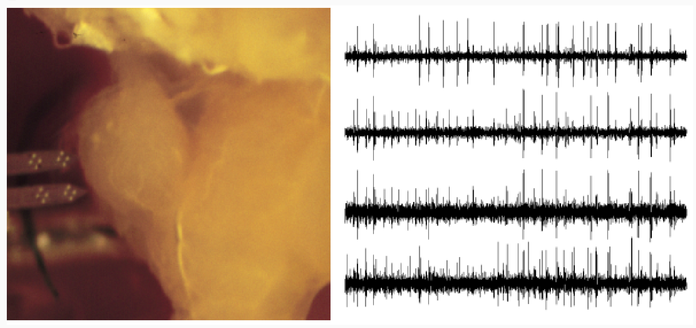
Christophe Pouzat
MAP5, Univ. Paris-Descartes and CNRS UMR 8145
Paris, France
christophe.pouzat@parisdescartes.fr

Georgios Is. Detorakis
LSS, Supélec, CNRS, Univ. Paris Sud
Gif-sur-Yvette, France
georgios.detorakis@lss.supelec.fr

Why do we need SPySort?

In Neuroscience, experimentalists use many different electrophysiological recording techniques to record neural population activities. One of the most commonly used methods is the extracellular recording technique giving *potentially* access to the activities of many identified neurons with a sub-millisecond resolution. Viewed "from the outside", neurons generate brief electrical pulses: the *action potentials*. Figure **1** shows a typical recording setting. Left, the brain of an insect with the recording probe on which 16 electrodes (the bright spots) have been etched. Each probe's branch has a 80 μm width. Right, 1 sec of data from 4 electrodes. The spikes are the action potentials. The raw data are a mixture of spikes coming from many neighboring neurons plus some recording noise. The first stage in the analysis of such data is the resolution of this mixture into its components, the signals of the individual neurons; a task referred to as **spike sorting** in the literature.

1



A spike sorting method

We assume, and check afterwards, that each neuron is characterized by a specific spike waveform (on each recording electrode) *that does not change*. That is, every time a given neuron fires a spike, the same waveform is "generated" and a *sampled* version of it with independent essentially Gaussian noise is observed / recorded. The algorithm implemented in SPySort was originally contributed by Pouzat, Mazor and Laurent (2002, Journal of neuroscience methods, 122, pp.43-57) and can be decomposed into the following steps:

- **Filtering / Rectification** - The raw data are filtered and amplitudes bellow a threshold are set to 0.
- **Spike detection** - Spikes are detected as local maxima on the rectified data.
- **Events extraction** - Events (waveforms) characterizing the observed spikes are cut around the detected maxima on normalized version of the data—normalization is done by dividing the raw amplitudes by the median absolute deviation (MAD), a robust estimator of the standard deviation of the noise.
- **"Clean" events extraction** - When many neurons are present in the recording, near coincident spike emission by two or more neurons is quite common. We want to identify such "superposed" events as such but in order to do that we need to have a good estimation of the waveform of each individual neuron. We therefore make a first "rough" identification of superposed events and remove them from the sample leading to a sample of "clean" events.
- **Feature extraction / Dimension reduction** - Principal Component Analysis (PCA) is applied. Dynamic multidimensional visualization with GGobi (`http://www.ggobi.org/`) is used to select the number of components to keep and to guess the number of neurons / clusters in the data.
- **Clustering** k-means or Gaussian Mixtures are used to estimate the center of each cluster. The result of this step is a catalog of waveforms: a waveform is associated to each identified neuron on each recording site.
- **Classification and alignment** We go back to the raw data and classify each detected event, resolving superposed events. A fast new method of sampling jitter estimation is used here.

SPySort is not an automated spike sorting software. On the contrary, it has been designed to be used in an interactive way . For instance, it can be used via **iPython**.

SPySort lives at:
`https://github.com/gdetor/SPySort`

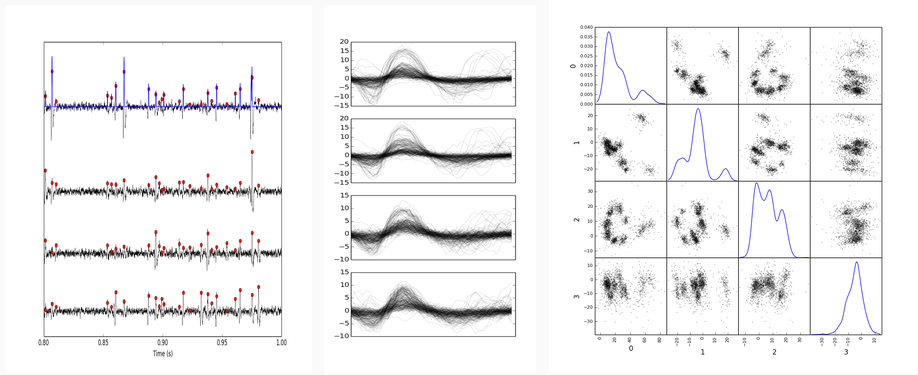
Structure of SPySort

SPySort is written entirely in Python. It exploits **Numpy**, **Scipy**, **Matplotlib**, **Pandas**, **Pickle**, **Scikit-learn** and **Sockets**. It consists in two main modules, one for reading binary data and one for processing the data implementing the aforementioned spike sorting method. A third module manages on-line transmission of the data to a server that, in turn, plots on-the-fly the results (if requested by the user).

Performing spike sorting ...

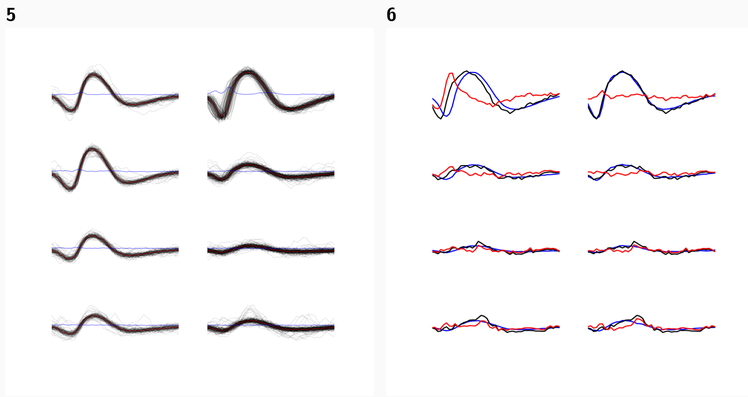
SPySort's illustration on the experimental data of Figure **1**. Raw data are read and normalized using the class **data** and its method **renormalization()**. SPySort can output the statistics of the original signals through a call to method **summary()**. Loaded data are then filtered and rectified and events are detected. This is done by importing module **Events** and using class **spike_detection**. Method **filtering()** filters and rectifies the data and method **peaks()** detects putative events. Figure **2** shows normalized data with candidate events (red dots) and the rectified version of the data on the first recording site (blue; detection is performed on the sum of the rectified data over the 4 recording sites).

2 **3** **4**



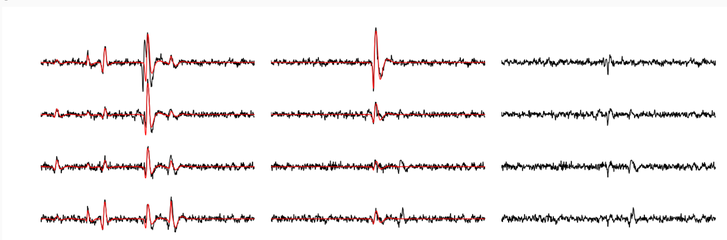
Events are then cut using method **mk_events()** of the **Events** module (Figure **3** shows the first 200 events, remark the presence superposed events, the horizontal axis spans 3 ms). Principal components are computed by calling method **pca_variance** of the **Clustering** module (Figure **4** shows the scatter plot matrix obtained from every possible pair of the first 4 principal components). Clustering is performed with methods **KMeans()** or **GaussianMixture()** of the same module. Figure **5** illustrates the clustering's results with the two clusters among ten (left and right column) giving the "largest" waveforms (red: median ; blue: MAD; same scales as Fig. **3**). A marked alignment jitter is observed on the first recording site for the events of the second cluster. The result of this clustering stage is a catalog of waveforms: a "central" waveform is associated to each cluster, together with its first and second derivatives.

5 **6**



After clustering, the **sampling jitter** is efficiently estimated and corrected for. The observed waveform $g(t)$ is assumed identical to a shifted version of the central waveform $f(t)$ plus some (nearly) Gaussian noise, giving: $g(t) = f(t+\delta) + \epsilon(t) \approx f(t) + \delta f'(t) + \delta^2/2 f''(t) + \epsilon(t)$, where δ is the jitter due to sampling and noise. δ is estimated via a 2 steps procedure: 1) neglecting the second order term, a first estimate is produced by the resulting linear regression problem; 2) the first estimate is used as a starting point for a one step Newton-Raphson algorithm including the second order term. Figure **6** illustrates the procedure (scales identical to Fig. **5**). On the left side: one event from the second cluster (black), the cluster central waveform (blue) and the difference of the two (red). On the right side: the same event (black), the second order correction (blue, $f(t) + \hat{\delta} f'(t) + \hat{\delta}^2/2 f''(t)$) and the difference of the two (red).
The complete classification of the data, including superposition resolution is done by a iterative "peeling" method: events are detected; the catalog's waveform providing the best fit is found—in the minimal residual sum of squares (RSS) sense, after time shifting for jitter compensation—; the RSS before and after subtraction of the best shifted waveform are compared and if the RSS decreases upon subtraction, subtraction is actually performed, otherwise the data are left (locally) untouched; the subtracted data are taken as new raw data and the procedure is repeated (keeping the same catalog); the procedure stops when no more subtractions are accepted. Figure **7** illustrates this procedure on the last 50 ms of Fig. **3**. Left: raw data (black) and first prediction (red); middle: previous raw data minus previous prediction (black) and new prediction (red); right: what's left (no more waveforms corresponding to the catalog's content).

7



Conclusions

SPySort is a spike sorting package written entirely in Python. It exploits **Numpy**, **Scipy**, **Matplotlib**, **Pandas** and **Scikit-learn**. It has been developed with simplicity and modularity in mind. SPySort can be easily integrated and cooperate with other data analysis packages. SpySort implements a thoroughly tested spike sorting method (Pouzat et al. 2002) **with major improvements concerning jitter estimation / correction and superposed events resolution**. SPySort is suitable for interactive data analysis using **iPython**.

Acknowledgments This work has been supported by the ANR JCJC project "Synch-Neuro".