RESEARCH PAPER

Acta Neurobiol Exp 2023, 83: 432–446
DOI: 10.55782/ane-2023-2440

Pylabianca: comprehensive and user-friendly Python package for single-neuron data analysis

Mikołaj Magnuski[1], Władysław Średniawa[1], Katarzyna Paluch[1], Davor Ivanovski[2], Harish Babu[1,2], Jan Kamiński[1]*

[1] Center of Excellence for Neural Plasticity and Brain Disorders: BRAINCITY, Nencki Institute of Experimental Biology, Polish Academy of Sciences, Warsaw, Poland,
[2] Department of Neurosurgery, SUNY Upstate Medical University, Syracuse, New York, USA,
** Email: j.kaminski@nencki.edu.pl*

In the area of electrophysiology, the availability of comprehensive and user-friendly tools for single neuron data processing, statistical analysis, and fast, intuitive data visualization is limited. To address this gap, we introduce pylabianca, a Python library tailored for robust single and multi-unit data processing. Pylabianca leverages the power of standard Python packages and adopts the application programming interface of MNE-Python, one of the most widely used electrophysiology packages. One of pylabianca's primary objectives is to provide a low entry threshold for scientists, requiring only basic Python programming skills. Pylabianca was designed to streamline most common analyses of single neuron data, and provide convenient data structures to serve as a foundation for building custom analysis pipelines. We believe that pylabianca will contribute to enhancing researchers' capabilities and efficiency in the field of single neuron electrophysiology.

Key words: python, toolbox, package, single-neuron, single-unit, analysis, decoding

INTRODUCTION

Rapid development of the electrophysiological methods allow us to monitor activity of hundreds of single neurons and spikes they produce simultaneously in different brain areas (Jun et al., 2017; Lehongre et al., 2022; Malach, 2021; Miccoli et al., 2019; Wang et al., 2023), bringing us closer to unravelling the complex mechanisms that underlie brain function. Across laboratories worldwide, researchers collect and analyze single-unit events from both animal and human subjects using various tasks and recording setup. By establishing connections between the firing rates of individual cells and experimentally controlled variables, researchers are able to construct models describing the transformation of sensory information from the receptor level to the cortex (Quiroga et al., 2005; Wiesel & Hubel, 1963). Single-unit studies contribute to an improved understanding of neuronal representation of the external world (Carvalho et al., 2020; Hafting et al., 2005; Quiroga et al., 2005; Sargolini et al., 2006; Solstad et al., 2008), or mechanisms of fundamental cognitive functions like working memory (Kamiński et al., 2017, 2020).

All this work requires an array of processing steps from the initial spike-sorting through multiple stages of analysis. However, a recurrent challenge arises from the diversity of analysis pipelines employed by different research teams, each implemented with custom code. This prevailing practice not only results in an abundance of redundant work reimplementing common functionality, but also introduces opportunity for error and inconsistency, impeding the comparability and replicability of findings, even within the context of identically sorted datasets. To address some of these challenges, a practical solution involves establishing a general-purpose library that exposes common data structures and

Received 25 October 2023, accepted 11 December 2023

operations. Such library, apart from providing standard analysis operations, could also serve as a foundation for building more complex and specialized analyses.

Within the Python ecosystem, the MNE-Python library is an example of such a successful effort at unifying continuous signal electrophysiology (Gramfort et al., 2013; 2014). Multiple specialized electrophysiology packages rely these days on MNE-Python: for example MNE-Connectivity (https://pypi.org/project/mne-connectivity/) and conpy (van Vliet et al., 2018) for connectivity estimation in source space, pactools (Dupré la Tour et al., 2017) for phase-amplitude coupling, mne-rsa (https://users.aalto.fi/~vanvlm1/mne-rsa/) for representational similarity analysis or MNELAB (https://mnelab.readthedocs.io/en/latest/), providing a graphical user interface similar to EEGLAB.

Over the years, the Application Programming Interface (API) of MNE-Python has matured, offering an excellent framework for other packages to emulate. While MNE-Python primarily supports the analysis of continuous signals rather than discrete spike trains, our objective is to broaden its functionality to spike analysis. To this end we developed pylabianca, a spike train analysis package that adheres to the elegant API design of MNE-Python. Guided by several principles, the development of pylabianca focused on: 1) maintaining object-oriented interface that offers a small number of objects for efficiently representing and manipulating spike train data; 2) allowing users to transition from reading the data to statistical analysis and visualization in a few simple steps; 3) utilizing labeled arrays (xarray DataArrays) where possible. These principles collectively aim to minimize the software's complexity, lowering the entry bar for new users. Object oriented data containers hide unnecessary intricacies while presenting processing options concisely and making them more discoverable through the exposed methods. Establishing a fast route to statistical inference requires an intuitive approach to handling, selecting and comparing experiment metadata (such as task conditions or subject responses). Lastly, the use of richer array representation results in self-documenting output, inheriting all the necessary experiment metadata for subsequent analysis steps.

Other dedicated Python packages for single-unit data analysis already exist, but they are either focused only on spike sorting (e.g., spike-interface, Buccino et al., 2020, SpyKING CIRCUS, Yger et al., 2018) or lack a high-level API for working with epoched data, condition selection, and statistical evaluation (elephant, Denker et al., 2018; spiketools, Donoghue et al., 2023) that pylabianca provides.

A recent python package, pynapple (Viejo et al., 2023), shares some common goals with pylabianca: pynapple aims to provide a concise set of domain-general objects capable of representing various data streams in neuroscience, including spike trains. However, there are noteworthy distinctions between pylabianca and pynapple. Pynapple adopts a more abstract approach, offering general tools for discrete event analysis, while pylabianca is specifically tailored for single-unit analysis. Consequently, some operations such as plotting spike waveforms, raster plots, or firing rates are more straightforward in pylabianca. Moreover, while pynapple employs its own objects for continuous signals, we integrate with other popular packages such as MNE-Python and xarray. By utilizing tools that many users may already be familiar with, pylabianca becomes more accessible and easier to maintain. Finally, pynapple does not currently provide statistical analysis in the form of cluster-based permutation tests or handling of experiment metadata comparable to pylabianca.

Below we provide a tutorial-like overview of pylabianca functionality using example human intracranial recording (data shared by our lab as a sample dataset for pylabianca, see Methods section) and monkey intracranial data provided in the Fieldtrip spike train analysis tutorial (see details in the Methods section). The examples presented here can also be viewed and followed interactively by downloading jupyter notebooks from pylabianca github repository (https://github.com/labianca/pylabianca/tree/main/doc).

METHODS

Software

We used Python 3.10.11 and pylabianca version 0.2 (https://github.com/labianca/pylabianca) installed from Python Package Index (PyPI) using a standard pip command (`pip install pylabianca`). All other packages used (pylabianca obligatory and optional dependencies) are listed in Table 1. The exact environment used for this publication can be recreated using Conda package manager form environment_ANE.yml file provided on pylabianca github page.

Primate intracranial recording

In some of the examples presented here we use monkey intracranial data collected by Pascal Fries and colleagues (2001, 2008), carrying out analyses on one sample file shared on the FieldTrip spike train analysis online tutorial (Oostenveld et al., 2011). Here we provide an overall description of the task and the recording. For further details see (Fries et al., 2008). The task of the monkey was to focus attention on one of

Table 1. Table characterizing python packages and their versions used for the examples.

Package	Functionality	Dependency relation to pylabianca	Version
numpy	operations on numerical arrays	required	1.24.4
pandas	operations on tables (DataFrames)	required	2.1.0
matplotlib	visualization	required for plotting	3.8.0
xarray	operations and visualization of labelled arrays	required	2023.8.0
scipy	used by pylabianca for common signal processing steps like filtering or convolution	required	1.11.2
mne-python	continuous signal electrophysiology	required	1.5.1
borsar	extensions to mne-python, particularly for cluster-based tests	required	0.1
tqdm	progress bar display	optional	4.66.1
scikit-learn	machine learning	optional	1.3.0
joblib	parallel processing	optional	1.3.2
numba	just in time (JIT) compilation to speed up computations	optional	0.57.1
neo	reading various neurophysiology formats	optional	0.12.0

two stimulus locations, wait for the stimuli to appear and then report change in the stimulus color in the attended location, ignoring color change of the stimulus in the ignored location. The monkey was rewarded with drops of apple juice for correctly reporting color change between 150 and 650 ms after it happened. While monkeys performed the task, electrophysiological signal (local field potentials and Multi unit activity) was recorded from the V4 brain area. The stimuli locations were set so that one of the locations overlapped with mapped receptive fields (RF) of V4 neurons and in a given trial, the stimulus in the RF was either attended (attention inside the RF) or not attended (distractor; attention outside the RF). In the examples provided in this paper, we analyze the spike rate data around the presentation of the stimulus, contrasting correct and incorrect trials or attention in RF *vs.* attention outside RF (only for correct trials).

Human intracranial recording

Other examples use human intracranial data recorded by our group. These data were obtained from a patient who underwent epilepsy monitoring with implanted depth electrodes for localization of the seizure focus as part of their presurgical plan for resection. The patient volunteered to participate in the study and gave informed consent. The study was approved by the SUNY Upstate Medical University Institutional Review Board, and all patients provided written informed consent. Electrode localization and surgery was based on clinical need and criteria only. The surgery was performed in SUNY Upstate Medical University Hospital, Syracuse, New York. Data contains recordings from three Behnke-Fried electrodes implanted into the medial temporal lobe (MTL). Each electrode contained eight macro contacts and eight 40-μm diameter microwires. We recorded broadband (0.1–9,000 Hz filter) data sampled at 32 kHz from the microwires using a Neuralynx Atlas system. Signal from each microwire bundle was locally referenced to one of the microwires. To extract single-units spikes, the data was filtered in the 300-3,000 Hz using a zero-phase lag filter, and OSort, a semiautomated template-matching algorithm (Rutishauser et al., 2006) was used. As a result of local referencing, units close to the reference microwire can be seen on all other microwires from the same bundle. To detect such cases we computed the coincidence of spikes produced by all pairs of units recorded from a given microelectrode bundle. Clusters of unit pairs with spike coincidence ≥30% were considered as duplicates – only one of these units was selected for further analysis.

The task performed by the patient is a modified Sternberg task (1966) – in each trial a sequence of 2-3 images is presented on a laptop screen and the goal is to remember the identity and order of these images. The patient maintains the images in memory for 10.5 seconds after which a digit query is presented – a number indicating the sequence position of the image to recall. After 2 seconds a probe image is shown (while the

digit query is still present on the screen but in a minimized form above the image). The participant then has to report, by pressing a button on the response box, whether the probe image was in the queried position in the sequence or not. This procedure was preceded by a screening session used to select images evoking good image-selective responses in the recorded single units (more detail can be found in Kamiński et al., 2017).

Reading the data

To read both the human and primate intracranial data we use pylabinaca.io submodule. The import convention of pylabianca is `import pylabianca as pln`, so the `io` submodule is accessed as `pln.io`. The human data are in OSort spike sorter format, so we use `pln.io.read_osort()` to read it (Fig. 1, code snippet on the left), while the primate data are in Plexon NeuroExplorer format – for which we use `pln.io.read_plexon_nex()`. We showcase reading these two formats of spiking data, but pylabianca also allows to read data stored in FieldTrip format (.mat files with a matlab structure following fieldtrip spike and spikeTrials convention), combinato (structure of folders with HDF5 files) or many other formats used by spike sorters *via* python neo package.

Raw spike data are read in pylabianca as a `Spikes` object. `Spikes` is one of two central data containers allowing for many diagnostic plots and processing options, both inspired with how FieldTrip (Oostenveld et al., 2011) organizes spike data. The data structure of `Spikes` is shown in Fig. 1. For the rest of this paper we will assume that the `Spikes` object resulting from reading the human example data is stored in a `spk_hum` variable, while the monkey data – in a `spk_mnk` variable. Spikes object instance contains two obligatory attributes: `timestamps` and `sfreq`. `timestamps` attribute contains a list of arrays containing timestamps when spikes occurred. The list is the same length as the number of single cells that spikes were recorded from, so `spk_hum.timestamps[0]` returns all the spike timestamps for the first cell (Python indexing is 0-based), while `spk_hum.timestamps[4][:100]` returns the first 100 spike timestamps for the fifth cell. The `sfreq` attribute contains a floating point value describing the sampling frequency of the timestamps.

`Spikes` instance will often contain (but does not have to) additional attributes containing for example cell names (`cell_names` attribute containing a list of strings), waveforms for individuals spikes (`waveform` attribute, read only when `waveform=True` argument of appropriate reading function is set) or a table with information about each cell (`cellinfo` containing pandas DataFrame). For a more detailed description of these attributes see Fig. 1.

Diagnostic plots and preprocessing

Pylabianca `Spikes` objects (but also `SpikeEpochs`, introduced later on) provide multiple methods for visually assessing single unit quality and performing simple preprocessing. Two common approaches to check quality of recorded single neurons are Inter-Spike Interval (ISI) and spike waveform plots. The first visualizes the histogram of time differences between consecutive spikes of a given cell (Hill et al., 2011) – a good single neuron should exhibit a drop in ISI frequency below 2-3 milliseconds corresponding to the refractory period. To plot an ISI histogram of units, we use the `.plot_isi()` method of the `Spikes` object (Fig. 2A, B for examples). The third plot visualizes action potential waveforms for a given unit. In pylabianca the waveform plots are implemented as two-dimensional histograms, in essence displaying the density of spike waveforms – this type of visualization allows for easier assessment of waveform quality by eye (see examples in Fig. 2, panel C). To plot the waveforms of one or multiple units, we use the `.plot_waveform()` method of the `Spikes` object. Both of these plots are also available through functions in the `viz` submodule of pylabianca: `pln.viz.plot_isi()` and `pln.viz.plot_waveform()`. Auto- and cross-correlation histograms can also be plotted using `pln.spike_distance.xcorr_hist()` (see Fig. 2, panel D).

Pylabianca also implements some simple preprocessing steps: peak-realigning waveforms and rejecting spikes corresponding to improbable waveform shapes (`pln.utils.realign_waveforms`), selecting or removing cells (`.pick_cells()` and `.drop_cells()` methods of `Spikes` and `SpikeEpochs`), merging units together (`.merge()` method of `Spikes`), adding brain anatomical information to `cellinfo` attribute or sorting the cells by some property added to the `cellinfo` table.

Epoching

To analyze changes in spiking activity related to some event of interest, for example stimulus onset, we first epoch the data using the `.epoch()` method of the `Spike` object. Epoching is a process of selecting data that are no further away from the event of interest than a specified time range – and in the context of spikes, changing their time representation from absolute timestamps to time distance to the event of interest. To epoch the data we

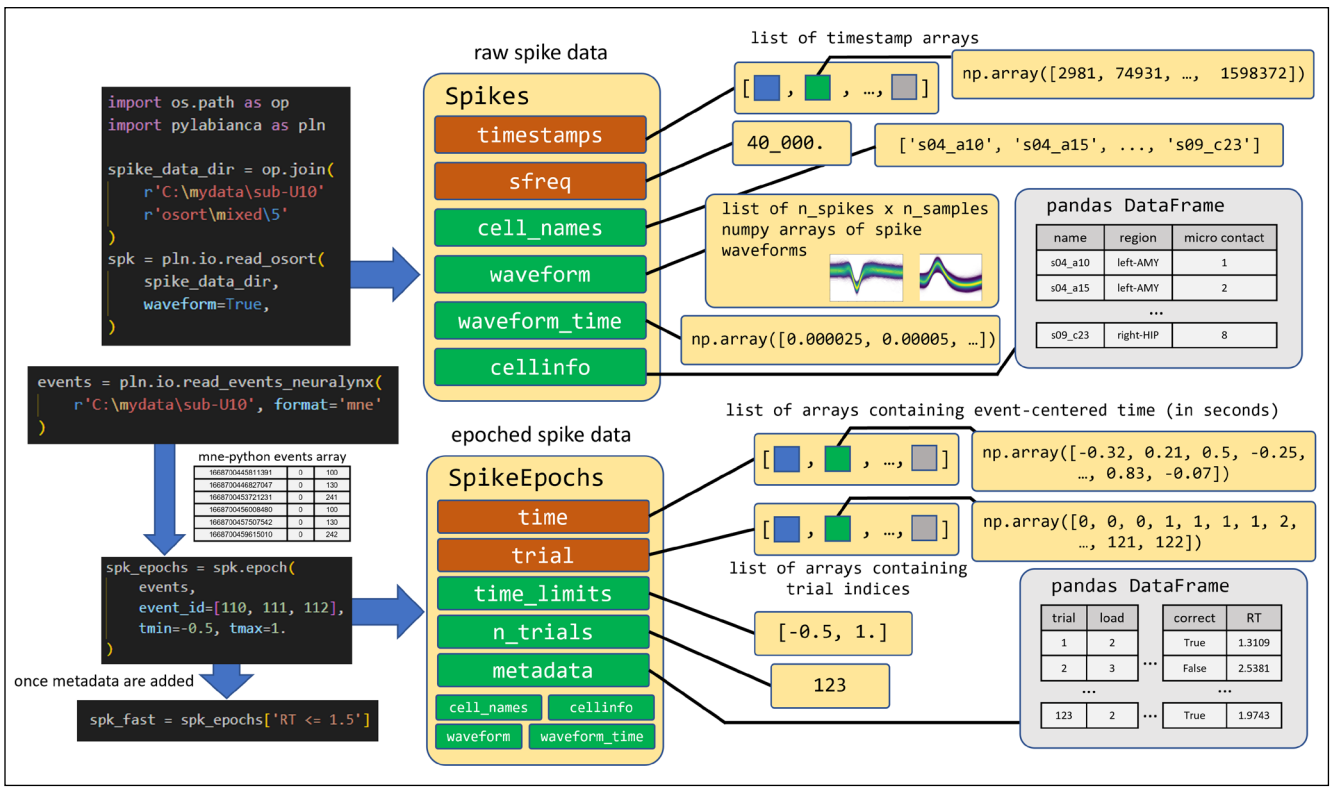
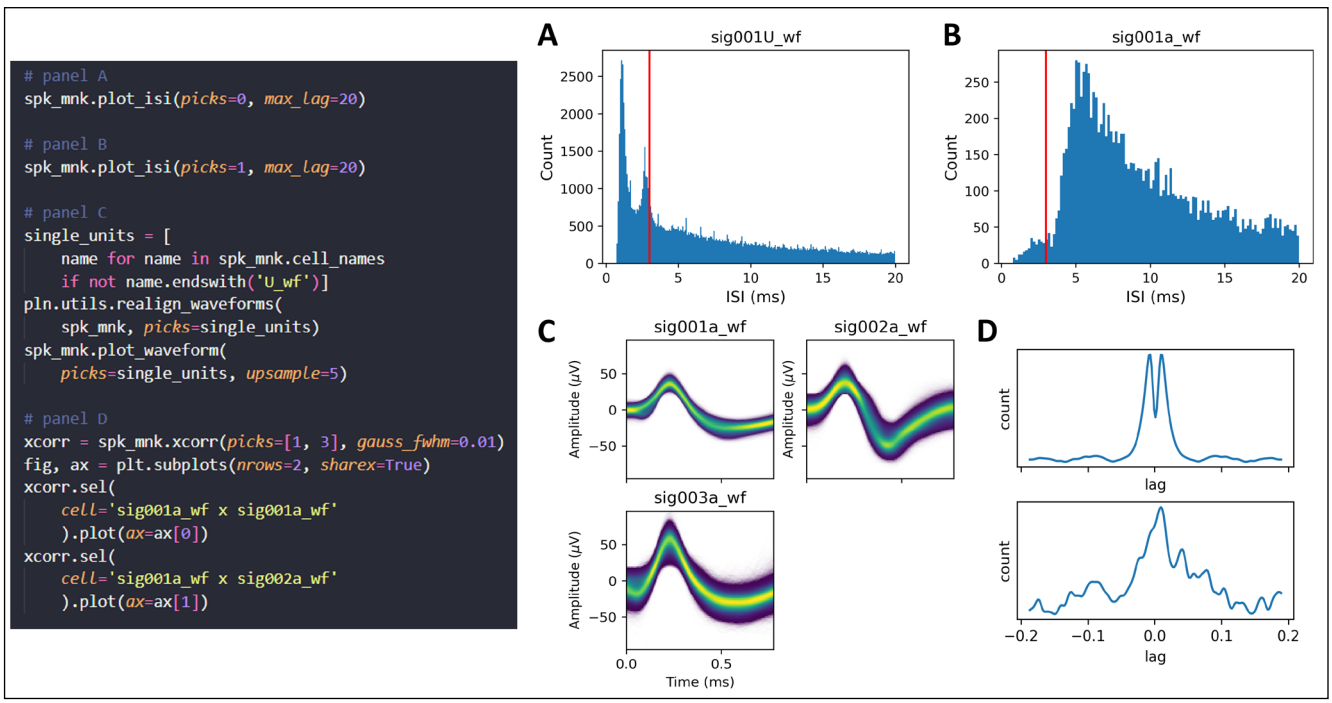


Fig. 1. Data organization in two fundamental pylabianca objects: `Spikes` and `SpikeEpochs`. The left side of the figure shows example code snippets for reading and epoching the data, while the right side shows the data containers that are returned. Within each data container – smaller boxes represent attributes – orange boxes mark required, while green boxes – optional attributes. Reading the output of a spike sorter returns a `Spikes` object, which contains raw spike information – that is, spike timestamps. `Spikes` attributes are shown on the top-right side of the figure: `timestamps` is a list of timestamp arrays (one list element corresponds to one neuron and its spike timestamps), `sfreq` is a floating point number describing sampling frequency of the timestamps, `cell_names` is a list (or array) of cell names, `waveform` – list of arrays, each containing single spike waveforms for respective neuron, `waveform_time` – time axis coordinates for the spike waveforms, `cellinfo` – cells x columns pandas DataFrame of arbitrary cell-level information (can contain channel id, channel type, brain location etc.). Epoching the data using `Spikes .epoch()` method creates a SpikeEpochs object instance. This object also contains spike times but now in seconds with respect to epoching events of interest – this information is stored in the `time` attribute as a list of arrays (one timing array per neuron) Each spike time from `time` attribute has a corresponding integer value in `trial` attribute that specifies the epoch index given spike belongs to. So `spk_hum.time[3][9:15]` – times of from 10th to 15th spike of fourth neuron have their corresponding trial indices in `spk_hum.trial[3][9:15]`. All pylabianca analysis functions respect the fact that spikes come from different trials and perform their computations accordingly. Two next attributes are created automatically during epoching: `time_limits` corresponds to `tmin` and `tmax` arguments passed to `.epoch()` method and `n_trials` stores the information on how many trials were there in total. The `metadata` attribute allows to store all experiment and behavioral information in a pandas DataFrame (trials × columns). This `metadata` attribute is also optional but can be very useful to add as it allows to select trials based on experiment conditions (see section "Experiment metadata and condition management").

first need to read information about the timing of the events: the human dataset has this information stored in a separate Neuralynx file – Events.nev and we read it using `pln.io.read_events_neuralynx` function; while for the monkey dataset, the events are in the same file as the spiking data – we use `pln.io.read_events_plexon_nex`. The default representation of event identity and timing in pylabianca is the same as MNE-Python – a numpy array of shape n_events by 3, where the first column represents event time, in timestamps, while the last column – numerical event identity (the middle column is not used but kept for compatibility with mne). We epoch the data with respect to presentation of each image to memorize in the human data; and with respect to stimulus presentation for the monkey data. For both datasets we set the lower time limit, controlled by the `tmin` argument, to -0.5 (0.5 seconds before event onset) and the upper time limit, controlled by the `tmax` argument, is set in the following way: 2 seconds for the human data and 1 second for the monkey data (the monkey data contain V4 neurons with fast responses to stimuli, while the human data contains units from medial temporal lobe, where responses to images are typically no earlier than 300-400 ms).




```
# panel A
spk_mnk.plot_isi(picks=0, max_lag=20)

# panel B
spk_mnk.plot_isi(picks=1, max_lag=20)

# panel C
single_units = [
    name for name in spk_mnk.cell_names
    if not name.endswith('U_wf')]
pln.utils.realign_waveforms(
    spk_mnk, picks=single_units)
spk_mnk.plot_waveform(
    picks=single_units, upsample=5)

# panel D
xcorr = spk_mnk.xcorr(picks=[1, 3], gauss_fwhm=0.01)
fig, ax = plt.subplots(nrows=2, sharex=True)
xcorr.sel(
    cell='sig001a_wf x sig001a_wf'
    ).plot(ax=ax[0])
xcorr.sel(
    cell='sig001a_wf x sig002a_wf'
    ).plot(ax=ax[1])
```

Fig. 2. Example diagnostic plots created with pylabianca. The left side of the figure shows a code snippet used to create the three plots on the right. (A) inter-spike interval histogram created by calling `.plot_isi` method. `picks=0` argument is used, so the histogram is plotted only for one unit. The histogram manifests some violations of the refractory period, suggesting that this unit is in fact a multi-unit. (B) Same as A, but for another unit (`picks=1`), which does not manifest clear violations of the refractory period. (C) Waveform density plots were created only for the units verified by the ISI histogram to be single cells. These cells' original names end with "U_wf", so we select them by name (`single_units` variable, "# panel C" section of the code snippet) Then, after peak-aligning the waveforms for these units, we plot the realigned waveforms using the `.plot_waveform()` method. (D) Example auto- and cross-correlogram computed with pylabianca.

The result of epoching in pylabianca is an `SpikeEpochs` object instance – the second of the two fundamental data structures used in pylabianca to represent spiking data. `SpikeEpochs` stores spike times in seconds with respect to selected epoching events. This information is contained in the `time` argument – a list of numpy arrays of spike times. Because these spike times, stored in one array, now can appear in different trials, each of these spikes has a corresponding trial index stored in `trial` attribute. See lower part of Fig. 1 for a graphical explanation of `SpikeEpochs` and a detailed description of its attributes.

Firing rate

Next, we estimate peri-stimulus time histogram both using standard sliding rectangular window approach and by calculating spike density – that is by convolving a binned representation of the spikes with a gaussian window. Both approaches are implemented in pylabianca as methods of `SpikeEpochs` object. Rectangular spiking rate is calculated using `.spike_rate()` method and parameterized with window length (`winlen`) and steps size when sliding the window across time (`step`). Spike density, calculated using `.spike_density()` method is implemented in two steps: first the spiking data are transformed to binned format (peri-stimulus spike histogram) with bin width defined as sampling frequency (`sfreq` argument, defaults to 500 Hz); then the binned representation is convolved with a gaussian window whose width can be defined both in terms of standard deviation (`gauss_sd`) or full width at half-maximum (`fwhm`).

The output of both methods is an xarray DataArray of size cells x trials x time. Because xarray is an array format that stores data along with dimension names and coordinates, the returned arrays are self-documenting and intuitive to work with in terms of manipulation and plotting. We showcase some of the benefits of this array format in Fig. 3: each of the subplot panels in the figure is generated with a simple line of code. Additionally, because xarray currently does not allow for drawing line plots with error intervals, we expose `pylabianca.viz.plot_shaded`, a function for creating such plots from xarray DataArrays – with standard error of the mean intervals shaded around the lines representing the average (Fig. 3D and Fig. 4B-D).

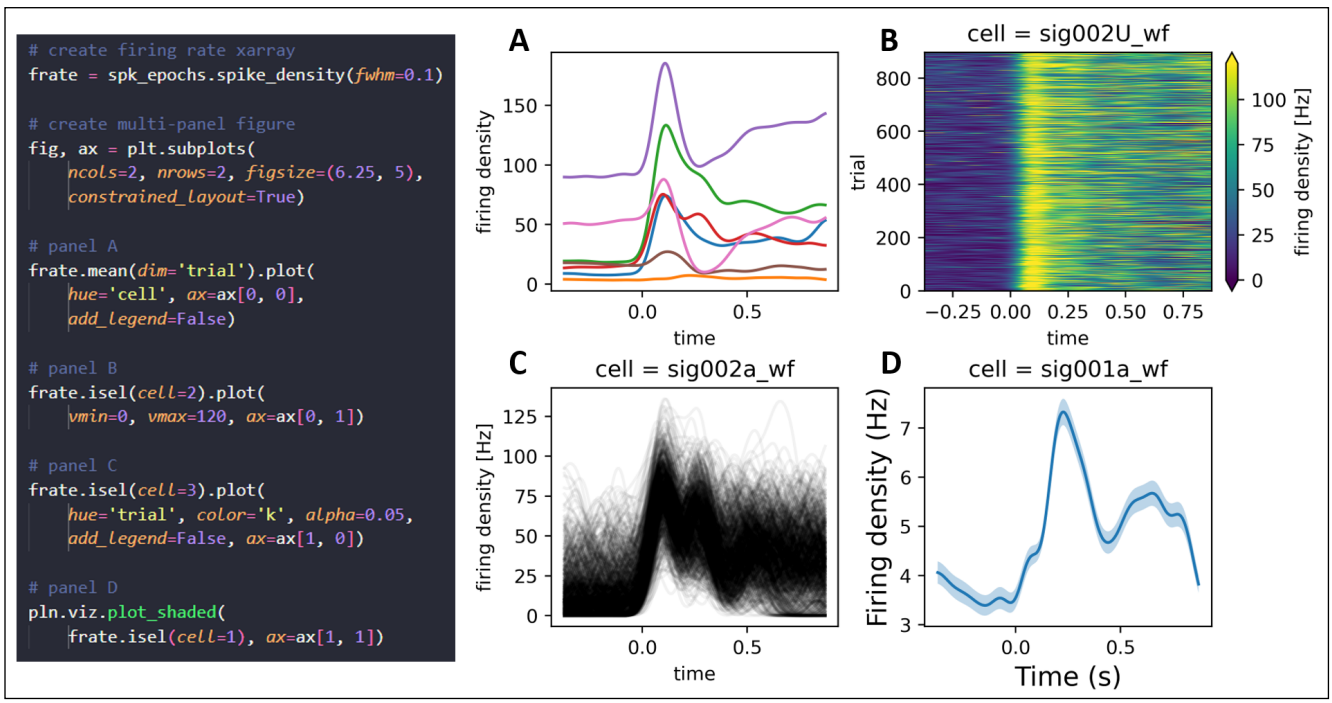


Fig. 3. Calculating and visualizing firing rate with pylabianca and xarray. The left side of the figure shows a short code snippet needed to generate the plots displayed on the right side. First, spike density is calculated for all the units and trials. The resulting `frate` xarray is then used to plot: (A) average spike density for all units; (B) single-trial spike density for the third cell using a heatmap; (C) single-trial spike density for the fourth cell with separate lines; (D) `pln.viz.plot_shaded` is used to plot peri-stimulus time histogram of second cell with shaded intervals representing standard error of the mean.

Firing of a neuron can also be displayed in an unaggregated way with a raster plot – in pylabianca this can be done with `pln.viz.plot_raster()` function. This function does not rely on xarray DataArrays – it takes SpikeEpochs object as its first input, but similarly to `pln.viz.plot_shaded()` – it allows to plot condition levels with different colors with `groupby` keyword argument (see examples in Fig. 4, panels A and B). Adding condition information to SpikeEpochs is covered in the next section.

Experiment metadata and condition management

Similar to mne-python, pylabianca allows adding a metadata table to the epoched data format (SpikeEpochs). Such a metadata table represents additional experiment per-trial information like condition variables, or behavioral outcomes of a trial (reaction time, correctness etc.). In pylabianca this table is a pandas DataFrame (the pandas development team., 2020) with the same number of rows as there are trials in SpikeEpochs and an arbitrary number of columns. These columns can be later referenced by their name to simplify trial selection. For the human data we add the metadata by reading a csv file generated by the experiment, while for the monkey data we create such metadata table based on the event sequences read from the .nex file (the code for creating this table is available online: https://github.com/labianca/pylabianca/blob/main/pylabianca/scripts/manipulate_fieldtrip_events.py).

Once the metadata table is added to SpikeEpochs, additional trial management options become available. This includes convenient trial selection by indexing the SpikeEpochs instance with a pandas query string: for example selecting specific experiment condition, like memory load (number of items stored in memory) can be simply done by writing `spk_load3 = spk_epochs['load == 3']`; or retaining trials with sufficient performance of the participant, like fast reaction time and correct response would be `spk_good = spk_epochs['RT<2. & correct == True']`. By sub-selecting trials in this manner, the relevant metadata table rows are retained, relieving the researcher from the onerous task of book-keeping all trial-level experiment details. The added metadata is also inherited by xarray DataArrays produced as a result of pylabianca function computing spike rate or cross-correlation – these metadata can then be used during plotting or statistical analysis (see examples in Fig. 4).

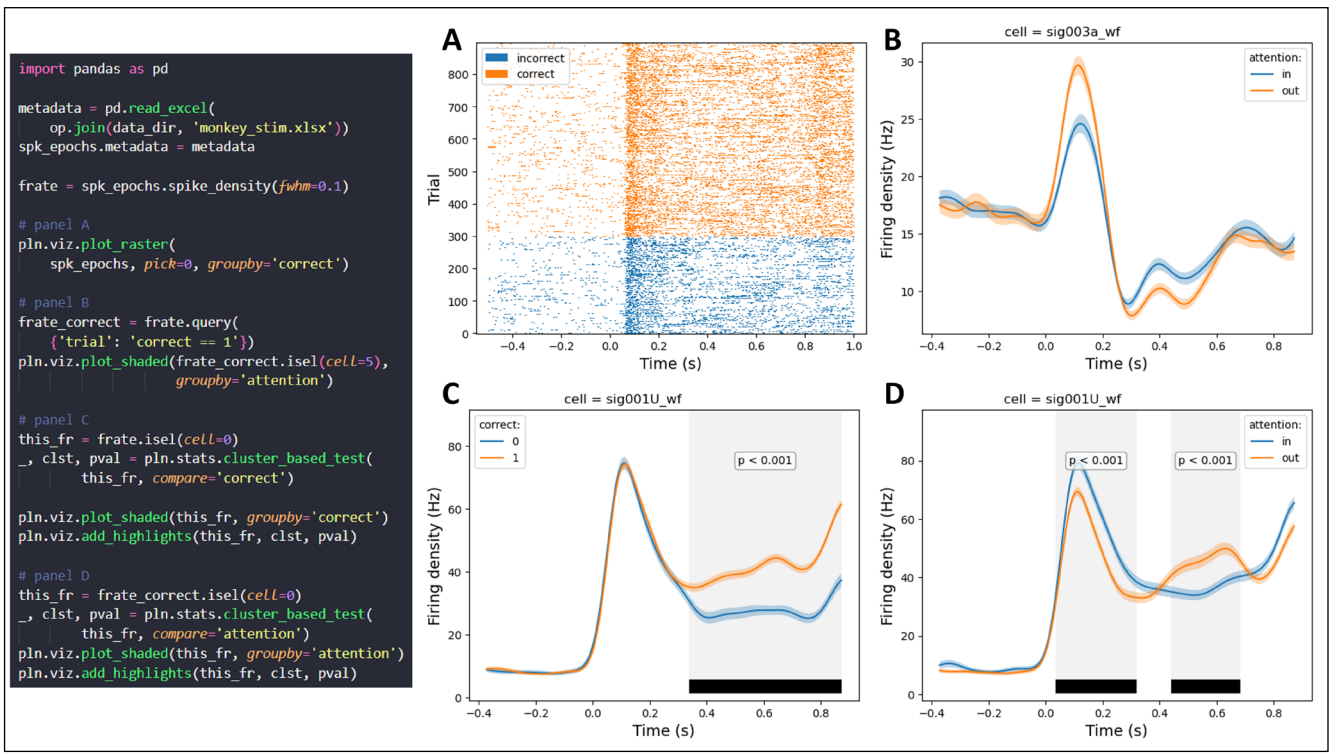


Fig. 4. Firing rate analysis and visualization. Lines representing average spike density were grouped and color-coded by condition levels. Shaded areas around the lines represent standard error of the mean. (A) Raster plot for the first cell (multi-unit) split by trials ending with correct *vs.* incorrect response. Each row is a single trial and each dot represents an action potential ; (B) peri-stimulus spike density plot for the sixth unit computed only for correct trials. The lines represent averages for separate attention conditions. (C) comparison of correct and incorrect firing density averages for the first unit (multi-unit) Black bar at the bottom and gray background highlight cluster ranges (here the cluster-level P<0.001) D) effects of attention in or away of the receptive field of neuron – tested on correct trials.

Cluster-based statistics

To test for the presence of a significant difference between conditions – for example image selectivity of a cell – we use cluster-based permutation test (Maris & Oostenveld, 2007). The core idea of this test is to compute statistics for continuous clusters passing a predefined threshold, not individual points in the data space. In the simplest case of time-resolved data, like firing rate of a cell or population of cells, the cluster-based test is performed by: a) first computing relevant statistical test (for example independent or repeated measures t-test or ANOVA) for each timepoint; b) defining adjacency clusters by finding continuous time ranges where the statistical test passes a predefined threshold (for example P<0.05 or specific t value); c) computing summary statistics by aggregating statistical test results for each cluster – a frequently used measure is the sum of individual test results. Once cluster-level summary statistics are calculated the significance is estimated *via* permutations: the condition structure is shuffled (permuted between observations for independent test or shuffled within some observations for repeated-measures test) and the steps a-c are repeated. The maximum/minimum cluster summary statistic is stored (depending on the statistical test performed, for example when using two-tail t test both minimum and maximum are stored, while for F test only the maximum is kept) at the end of each permutation. Repeating the permutations creates a null distribution for the cluster summary statistic. To get the p value, cluster summary statistics from the actual, non-permuted data are compared to the null distribution.

pylabianca relies on mne-python and borsar (https://github.com/mmagnuski/borsar) when performing the cluster based permutation test. The test can be performed and visualized in an easy, straightforward way – the type of statistical test can be inferred automatically and the condition structure is gathered from the xarray. This is for example how one would conduct and visualize the test comparing correct and incorrect trials:

```
frate_sel = frate.isel(cell=0)
stats, clusters, pval = pln.stats.cluster_based_test(
   frate_sel, compare='correct', paired=False)

pln.viz.plot_shaded(frate_sel, groupby='correct')
pln.viz.add_highlights(frate_sel, clusters, pval)
```

In the first line, the first neuron is selected by its index from the firing rate `frate` xarray. In the second line the cluster based test is performed using `pln.stats.cluster_based_test` passing the selected cell along with `compare` and `paired` arguments. The `compare` argument specifies the condition to perform comparisons for – this is the metadata column that the xarray DataArray inherited from `SpikeEpochs`. The `paired` argument specifies whether the condition type is independent (`paired=False`) or repeated (`paired=True`). Based on these two arguments the correct statistical test and permutation scheme is inferred. The next two lines visualize the results of the cluster-based test: `pln.viz.plot_shaded` is used to draw average firing rates with standard error of the mean grouped by the selected condition (`groupby='correct'`) and `pln.viz.add_highlights` marks time ranges with clusters corresponding to cluster-level p value<0.05 (this default can be changed by specifying `p_threshold` keyword argument). See Fig. 4, panels C and D for the resulting plot.

Condition selectivity

pylabianca enables users to search and test for condition selectivity of single units – `pln.selectivity` module provides multiple functions to achieve this. One of these functions, `pln.selectivity.cluster_based_selectivity`, performs a cluster-based test on each of the cells to detect those that differentiate levels of the condition of interest. Here, we will demonstrate this functionality in the context of image selectivity of medial temporal lobe neurons. We first perform a search for cells selective to image identity for all the images that appear on the first position in the sequence, and then to validate their selectivity we perform another cluster based test on a separate part of the data. This second test is restricted only to the units selected by the first test and time segments corresponding to presentation of the images on the second position in the sequence (see Fig. 5 for the full code and plot examples). Searching for selective cells with `pln.selectivity.cluster_based_selectivity` returns a pandas DataFrame with one row per detected cluster (irrespective of cluster-level p value) containing cell name, brain region (if it was provided in "region" column in SpikeEpochs.cellinfo table – and later inherited by firing rate xarray), cluster-level p value, cluster timespan, preferred condition levels (multiple levels can be specified if they evoke comparable firing rate to the preferred one), depth of selectivity (Rainer et al., 1998) or explained variance (Jacob & Nieder, 2014; Okada, 2013) as well as average absolute and baseline-relative firing rate to the preferred category. These results can be then narrowed down to leave only units with good selectivity using `pln.selectivity.assess_selectivity`. Rigorous selectivity criteria can be defined with `pln.selectivity.assess_selectivity` – for example minimum cluster timespan in window of interest (`window_of_interest` and `min_time_in_window` arguments) or minimum explained variance by the condition levels. An example unit surviving these steps can be seen in Fig. 5, panel A.

Decoding

Thanks to integration with scikit-learn, a powerful machine learning package, and mne-python, extending some of the scikit-learn functionalities to better suit electrophysiology data, pylabianca allows to perform decoding analyses in just a few lines of code. Decoding analyses are run with `pln.decoding.run_decoding()`, which, as many other pylabianca functions, takes an xarray DataArray as the first argument. The target to predict is specified with `target` argument, which indicates the name of the metadata column (inherited as a trial coordinate by the xarray DataArray). By default `pln.decoding.run_decoding()` performs classification "sliding" across time, that is the classification is done for each timepoint. To reduce the computation time the signal can be decimated (`decim` argument) and classification for separate points can be run in parallel (`n_jobs` controls the number of parallel workers). To perform generalization across time (King & Dehaene, 2014), where for each timepoint the trained classifiers are tested on all other timepoints, `time_generalization=True` can be specified. The default classifier used in `plt.decoding.run_decoding()` is a linear support vector machine, but any scikit-learn compatible classifier of classification pipeline can be passed to the function using the `clf` argument. pylabianca also provides an implementation of the maximum correlation classifier (maxCorr, Di Liberto et al., 2021) as `pln.decoding.maxCorr`. The output of `pln.decoding.run_decoding()` is an xarray DataArray with fold x time dimensions (where the "fold" dimension corresponds to results from successive train-test partitions in the k-fold approach) or fold x training time x test time for `time_generalization=True`.

To statistically evaluate the decoding results, one can construct a null distribution of decoding accuracy by permuting the relationship between condition labels and actual data many times. For each permutation the same decoding pipeline is run. This procedure is efficiently handled by the `pln.decoding.permute()` function (see Fig. 6 code snippet and panel D). To use

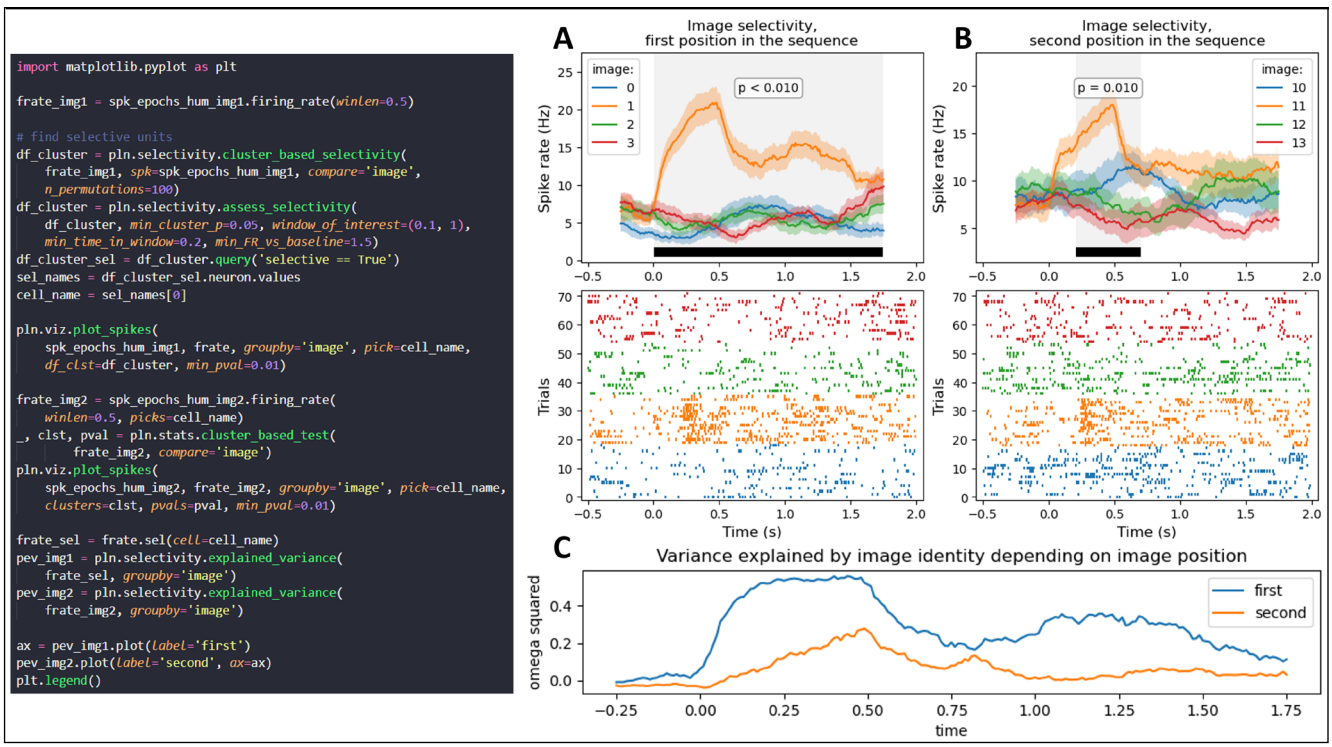


Fig. 5. Example of searching for selective units. After calculating the firing rate for all cells `pln.selectivity.cluster_based_selectivity()` is called to scan all units with a cluster-based test, evaluating their selectivity to image identity (for the first image in the sequence) `n_permutations=100` is used here for speed, but for real analysis that number should be higher. `pln.selectivity.asses_selectivity()` is then used to select units fulfilling selectivity criteria: cluster-level p value should be below 0.05 (`min_cluster_p=0.05`), the cluster timespan has to cover at least 0.2 seconds of 0.1-1 s time window (`window_of_interst=(0.1, 1)` and `min_time_in_window=0.2`). Additionally, the average firing rate within the cluster has to be at least 1.5 of the average firing rate in the prestimulus window of that cell (`min_FR_vs_baseline=1.57`). The first selective cell is plotted along with the timespan of the cluster and the raster plot (panel A) using `pln.viz.plot_pikes()`. Then to validate the selectivity of the cell we perform a cluster based test for this specific unit, but now using a separate data segment – presentation of the second image in the sequence (panel B). As the last, step we compare the omega squared percentage of explained variance by image identity (using `pln.selectivity.explained_variance()`) depending on the position on the image in the sequence (panel C).

this function, several parameters need to be specified: `target` variable; `decoding_fun` – the function used to perform the decoding, with `decoding_fun=pln.decoding.run_decoding` being the standard choice in our case; `arguments`: any additional arguments necessary for the decoding function; `n_permutations` – the number of permutations to generate; `n_jobs` – number of parallel processes to employ. The output of this process is an xarray with a structure closely resembling the one returned by `pln.decoding.run_decoding()`. However, an additional "perm" dimension is introduced: each element along this dimension represents the decoding results for different permutations of the target variable. This permutation array becomes a critical component in defining significance thresholds for each timepoint of the decoding. To account for multiple comparisons (given that each time point has a separate threshold for significance), you can use the `pln.stats.cluster_based_test_from_permutations()`. This function uses the permutation distribution to: a) specify a one-tail or two-tail threshold of significance (this can be controlled by the `tail` argument); b) conduct a cluster-based test to correct for multiple comparisons using the thresholds computed in the previous step. These two steps are semi-independent, as the first step focuses only on each of the time points separately, while the second one leverages continuous effects across time points by performing cluster-based correction for multiple comparisons. This approach is particularly valuable when dealing with measures that lack straightforward significance thresholds, such as decoding accuracy, the percentage of selective units or various connectivity measures (for example spike-field coupling). As a result, this approach finds broad applicability in single spike analysis, proving instrumental in rigorously evaluating effects that may deviate from the assumptions of classical statistical tests. Note, that this method can also be used in `pln.stats.cluster_based_test` by changing the `n_stat_permutations` argument.

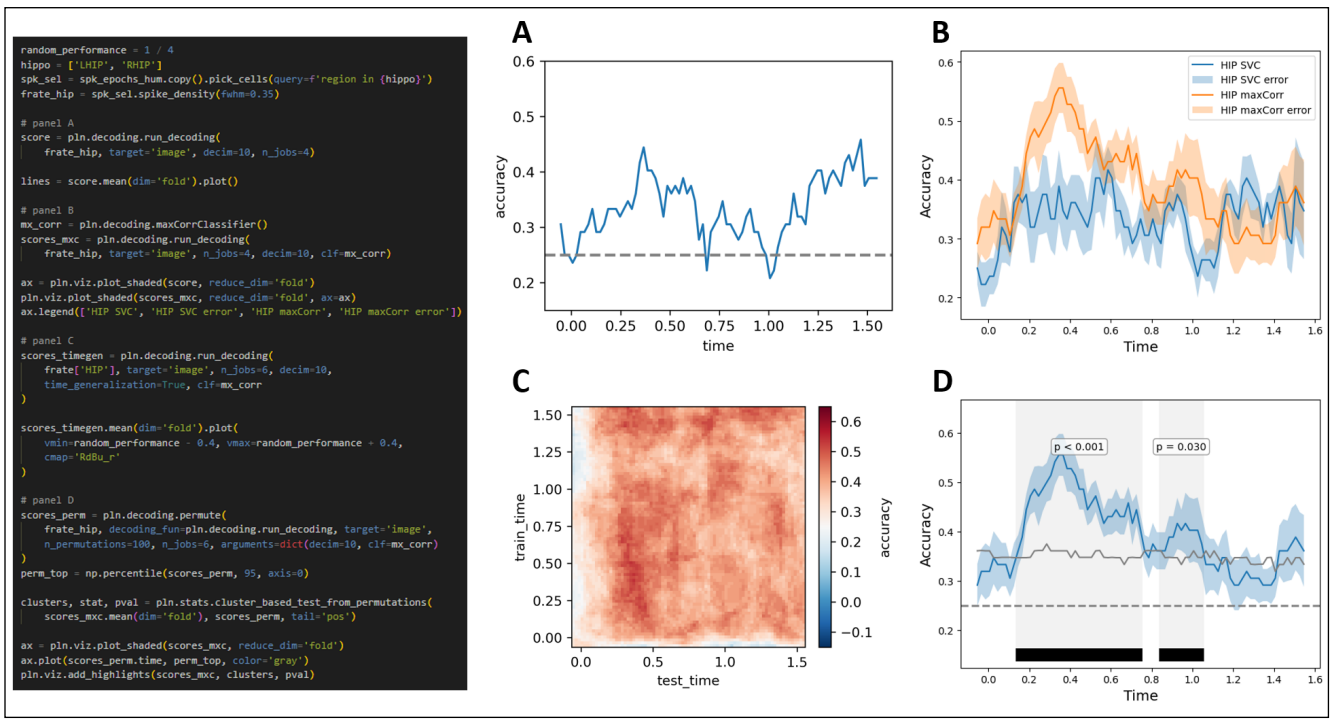


Fig. 6. Example of decoding analyses and visualizations. (A) Results of the `pln.decoding.run_decoding()` with default settings (scaling + linear Support Vector Machine classifier, default scikit-learn regularization) on the population of all hippocampal cells for the given patient. (B) Comparison of the default decoding accuracy *vs.* using pylabianca maxCorr implementation (`pln.decoding.maxCorr`) The maxCorr was used without wrapping it with scikit-learn Pipeline including standard scaling (see Results section for a discussion). The shaded areas around lines represent standard error of the mean across k train-test folds. (C) Results of using the same maxCorr classifier in the generalization across time approach. The classifier is trained for each time point (y axis) and then tested across all other timepoints (x axis). Accuracy is represented by a colormap centered at random performance (25%, white). (D) Testing decoding significance by applying permutations and cluster-based test to the maxCorr decoding. Gray line represents the top 5% of the permutation distribution (95th percentile) for each timepoint. Cluster ranges are marked with black bars and gray background. The corresponding cluster-level p values are shown in text boxes at the top.

Integration with other packages

Thanks to the integration with MNE-Python, which implements a host of functions for processing continuous signals, pylabianca can also be used for spike-field analysis: measuring relationship between spike times and features of the local field potential (LFP). A full example of such analysis performed with pylabianca and MNE-Python can be found online (https://github.com/labianca/pylabianca/blob/main/doc/spike-triggered_analysis.ipynb), here we will just summarize it briefly. We can identify and extract LFP segments centered on spike times of unit of interest using `pln.utils.spike_centered_windows()` which takes `SpikeEpochs` and `mne.Epochs` as inputs (although the second argument can be an xarray DataArray as well). The output is an xarray DataArray of size spikes x lfp_channels x time. This array can be averaged across spikes dimension to produce LFP spike-triggered average. We can compare this average LFP to what we would expect to get by randomly pairing trials from SpikeEpochs and LFP array. This can be achieved by first shuffling the `SpikeEpochs` trials using `pln.utils.shuffle_trials()` function before extracting spike-centered windows. The result of such comparison performed on example primate data can be seen in Fig. 7A. To further characterize the obtained spike-triggered oscillation we compute multitaper power spectral density using MNE-Python (using `mne.time_frequency.psd_array_multitaper`, Fig. 7B).

Pylabianca also provides functions to convert `SpikeEpochs` objects to representations used by other packages: elephant (`.to_neo()` method, converting to `neo.SpikeTrain`) or spiketools (`.to_spiketools()`, converting to lists of numpy arrays of spike times). This way pylabianca can be used to perform epoching and manage conditions and the analysis can be delegated to one of these packages. To streamline this kind of delegation, SpikeEpochs object exposes `.apply()` method that can be used to iteratively apply a function to multiple units and trials. An example of interacting with spiketools from pylabianca can be found here: https://github.com/labianca/pylabianca/blob/main/doc/working_with_spiketools.ipynb.

```python
channels = ['AD01', 'AD02', 'AD03', 'AD04']
lfp_poststim = lfp_epochs.copy().crop(tmin=0.3).pick(channels)
spk_poststim = spk_epochs.copy().crop(tmin=0.3)

# spike-triggered LFP for original data
lfp_trig = pln.utils.spike_centered_windows(
    spk_poststim, lfp_poststim, pick='sig002a_wf', winlen=0.4)
lfp_trig -= lfp_trig.mean(dim='time')

# spike-triggered LFP for trial-shuffled data
new_spk = pln.utils.shuffle_trials(spk_poststim)
lfp_trig_shuff = pln.utils.spike_centered_windows(
    new_spk, lfp_poststim, pick='sig002a_wf', winlen=0.4)
lfp_trig_shuff -= lfp_trig_shuff.mean(dim='time')

# panel A plot
ax = pln.viz.plot_shaded(lfp_trig.sel(channel='AD03'))
pln.viz.plot_shaded(lfp_trig_shuff.sel(channel='AD03'), ax=ax)

# compute multi-taper spectrum
args = dict(sfreq=lfp_poststim.info['sfreq'], bandwidth=7, fmin=10, fmax=100)
psd, freq = mne.time_frequency.psd_array_multitaper(
    lfp_trig.sel(channel='AD03').mean(dim='spike'), **args)
psd_shuff, freq = mne.time_frequency.psd_array_multitaper(
    lfp_trig_shuff.sel(channel='AD03').mean(dim='spike'), **args)

# panel B plot
fig, ax = plt.subplots()
ax.plot(freq, psd, label='original')
ax.plot(freq, psd_shuff, label='shuffled')
ax.set(xlabel='Frequency', ylabel='Power')
ax.legend()
```

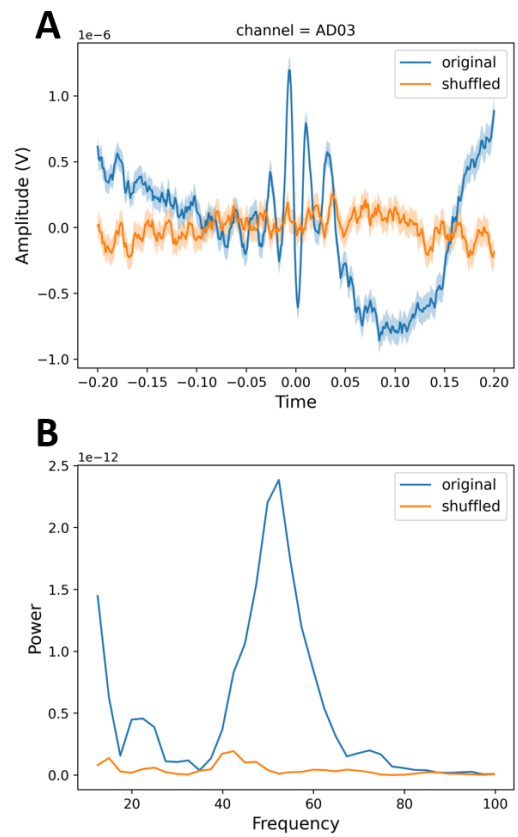


Fig. 7. Example of integrating pylabianca with MNE-Python to perform spike-triggered LFP analysis on the primate data. The full example can be found online (see text for link). (A) Spike-triggered LFP averages for original (blue) and trial-shuffled data (orange). (B) Multitaper spectrum of spike-triggered LFP averages.

RESULTS

To demonstrate the capabilities of the pylabianca package, we perform analyses on two example datasets, examining single unit responses in relation to various experimental conditions.

In the first part of the examples we use data from monkey recordings (Fries et al., 2001). First we visually examine superimposed spike waveforms and inter-spike interval (ISI) plots, as exemplified in Fig. 2. These visualizations allow to reveal groups of single and multi-units and assess their quality.

Comparing firing rates

We proceed by segmenting the data into epochs using the `.epoch()` method, and illustrate the process of generating firing rate (FR) plots for the units. Because both `.spike_rate()` and `spike_density()` methods return xarray DataArrays, we can take advantage of their versatile plotting options as depicted in Fig. 3. Pylabianca allows to attach the experiment metadata table describing task conditions to the `SpikeEpochs` object simplifying further data visualization and statistical testing. These metadata are inherited by xarray DataArrays produced by many pylabianca processing functions and methods. Thanks to this visually comparing conditions is as easy as adding a `groupby` keyword to respective plotting function. For example trials ending with correct *vs.* incorrect responses are compared on a raster plot (`.plot_raster()` method) and firing rate plot (`plot_shaded` function operating on xarray DataArrays) just by adding `groupby='correct'` argument (Fig. 4A and 4C). Similarly, we can statistically contrast the dynamics of unit responses across levels of attention condition by specifying `compare='attention'` argument in `pylabianca.stats.cluster_based_test()`. This function automatically determines the adequate statistical test to perform depending on the number of condition levels (this can be controlled by `stat_fun` parameter) and contrasts these conditions with correction for multiple comparisons by performing the cluster-based permutation test. The time rang-

es and p values of obtained clusters can be marked on the plot using `pylabianca.viz.add_highlights()` (although it is important to remember that technically the p values do not pertain to the cluster time ranges per se, but to the overall null hypothesis of condition exchangeability, see Sassenhagen and Draschkow 2019 or https://www.fieldtriptoolbox.org/faq/how_not_to_interpret_results_from_a_cluster-based_permutation_test/ for more information).

Testing for stimulus selectivity

We next present pylabianca data analysis of single units recorded in humans. We identify units selective to specific images with a cluster-based permutation test by first using `pylabianca.selectivity.cluster_based_selectivity()` and then refining selectivity criteria with `pylabianca.selectivity.assess_selectivity()`. Raster plots and average firing rate responses of one such selective unit are shown in Fig. 5. To validate selectivity of this unit, we perform the cluster-based test again – now using responses to the images that were shown on the second position in the sequence. The unit manifests selectivity to the same image in the second test, so it retains its selectivity from the first image position. To summarize the selectivity of the unit we can plot the percentage of explained variance in unit firing rate by image category for each timepoint using `pylabianca.selectivity.explained_variance()` (Fig. 5C).

Running a decoding analysis

In Fig. 6 we present example decoding analyses aimed at inferring the identity of presented images from the entire population of hippocampal cells. A decoding analysis can be easily performed with `pylabianca.decoding.run_decoding()` passing an xarray DataArray (firing rate) and the name of the metadata column to use as target in the decoding (`target='image'`). To begin with we compare decoding accuracy obtained when using the default Support Vector Machine (SVM) and the maxCorr classifier. The maxCorr classifier is available in pylabianca, but any classifier or sklearn Pipeline can be passed to the decoding function.

Finally, to statistically evaluate the decoding results we perform a cluster-based permutation test. Because this is a less common application of the cluster-based test (bear in mind that we don't have repetitions like multiple trials, cells or participants here) it is not handled by `pln.stats.cluster_based_test()`. Instead, we run the permutations ourselves with `pln.decoding.permute()`, shuffling the decoding target in each permutation, and then pass the xarray DataArray of permutation outcomes to more specialized `pln.stats.cluster_based_test_from_permutations()`. The results of the decoding analysis can be then visualized using the same set of functions that were used to plot firing rate results (compare code snippet for Fig. 4C and 4D with that for Fig. 6D).

DISCUSSION

Pylabianca is a software package developed to streamline the processing of single unit data. It provides a straightforward Application Programming Interface inspired by MNE-Python and utilizes a data storage structure similar to FieldTrip. Like MNE-Python, pylabianca offers the ability to include trial-level metadata in the `SpikeEpochs` object simplifying subsequent condition selection and statistical analysis. It makes use of the xarray framework to store various outputs, such as firing rate, cross-correlation histograms, and decoding accuracy: these xarray DataArrays are equipped with dimension names and coordinates and inherit condition and cell metadata, making data manipulation and plotting more convenient. Pylabianca also simplifies the execution of cluster based tests and decoding analyses. We hope that by simplifying complex data analysis tasks and introducing rich visualizations, pylabianca will position itself as a valuable resource in the field of single-unit electrophysiology, offering a versatile toolkit to researchers.

In the ever-evolving landscape of scientific software, no software project is static, except for those that have been abandoned. We believe in the continuous improvement and evolution of pylabianca to ensure that it remains a valuable resource for the scientific community. As we move forward, the feedback and insights from pylabianca users will be at the core of our development efforts. In the near future we plan to extend pylabianca by exposing more utilities for spike-field analysis and providing better support for high-density probes by integrating with the ProbeInterface package (Garcia et al., 2022).

CONCLUSION

In conclusion, pylabianca is a user-friendly software package for efficient single-unit data processing in electrophysiology. It provides a straightforward API, integrates experiment metadata, and utilizes a robust data storage structure. The software simplifies com-

plex data analysis tasks, offers rich visualizations, and facilitates cluster-based tests and decoding analyses.

ACKNOWLEDGEMENTS

This work has been financially supported by the Polish National Science Centre NCN Grant: 2019/34/E/HS6/00257 and by the BRAINCITY MAB/2018/10. The "Nencki-EMBL Center of Excellence for Neural Plasticity and Brain Disorders: BRAINCITY" project is carried out within the International Research Agendas Programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

REFERENCES

Buccino AP, Hurwitz CL, Garcia S, Magland J, Siegle JH, Hurwitz R, Hennig MH (2020) SpikeInterface, a unified framework for spike sorting. eLife 9. https://doi.org/10.7554/eLife.61834.

Carvalho MM, Tanke N, Kropff E, Witter MP, Moser MB, Moser EI (2020) A brainstem locomotor circuit drives the activity of speed cells in the medial entorhinal cortex. Cell Rep 32: 108123.

Denker M, Yegenoglu A, Grün S (2018) Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework (FZJ-2018-04998). Computat Systems Neurosci. https://juser.fz-juelich.de/record/851308.

Di Liberto GM, Marion G, Shamma SA (2021) Accurate decoding of imagined and heard melodies. Front Neurosci 15: 673401.

Donoghue T, Maesta-Pereira S, Han CZ, Qasim SE, Jacobs J (2023) spiketools: a Python package for analyzing single-unit neural activity. J Open Source Software 8: 5268.

Dupré la Tour T, Tallot L, Grabot L, Doyère V, van Wassenhove V, Grenier Y, Gramfort A (2017) Non-linear auto-regressive models for cross-frequency coupling in neural time series. PLoS Computat Biol 13: e1005893.

Fries P, Reynolds JH, Rorie AE, Desimone R (2001) Modulation of oscillatory neuronal synchronization by selective visual attention. Science 291: 1560–1563.

Fries P, Womelsdorf T, Oostenveld R, Desimone R (2008) The effects of visual stimulation and selective visual attention on rhythmic neuronal synchronization in macaque area V4. J Neurosci 28: 4823–4835.

Garcia S, Guarino D, Jaillet F, Jennings T, Pröpper R, Rautenberg PL, Rodgers CC, Sobolev A, Wachtler T, Yger P, Davison AP (2014) Neo: an object model for handling electrophysiology data in multiple formats. Front Neuroinform 8: 10.

Garcia S, Sprenger J, Holtzman T, Buccino AP (2022) ProbeInterface: A unified framework for probe handling in extracellular electrophysiology. Front Neuroinform 16: 823056.

Gramfort A, Luessi M, Larson E, Engemann DA, Strohmeier D, Brodbeck C, Goj R, Jas M, Brooks T, Parkkonen L, Hämäläinen M (2013) MEG and EEG data analysis with MNE-Python. Front Neurosci 7: 267.

Gramfort A, Luessi M, Larson E, Engemann DA, Strohmeier D, Brodbeck C, Parkkonen L, Hämäläinen MS (2014) MNE software for processing MEG and EEG data. NeuroImage 86: 446–460.

Hafting T, Fyhn M, Molden S, Moser MB, Moser EI (2005) Microstructure of a spatial map in the entorhinal cortex. Nature 436: 801–806.

Hill DN, Mehta SB, Kleinfeld D (2011) Quality metrics to accompany spike sorting of extracellular signals. J Neurosci 31: 8699–8705.

Hoyer S, Hamman J (2017) xarray: N-D labeled arrays and datasets in Python. J Open Res Software 5: 10.

Jacob SN, Nieder A (2014) Complementary roles for primate frontal and parietal cortex in guarding working memory from distractor stimuli. Neuron 83: 226–237.

Jun JJ, Steinmetz NA, Siegle JH, Denman DJ, Bauza M, Barbarits B, Lee AK, Anastassiou CA, Andrei A, Aydın Ç, Barbic M, Blanche TJ, Bonin V, Couto J, et al. (2017) Fully integrated silicon probes for high-density recording of neural activity. Nature 551: 232–236.

Kamiński J, Brzezicka A, Mamelak AN, Rutishauser U (2020) Combined phase-rate coding by persistently active neurons as a mechanism for maintaining multiple items in working memory in humans. Neuron 106: 256–264.e3.

Kamiński J, Sullivan S, Chung JM, Ross IB, Mamelak AN, Rutishauser U (2017) Persistently active neurons in human medial frontal and medial temporal lobe support working memory. Nature Neurosci 20: 590–601.

King JR, Dehaene S (2014) Characterizing the dynamics of mental representations: the temporal generalization method. Trends Cognit Sci 18: 203–210.

Lehongre K, Lambrecq V, Whitmarsh S, Frazzini V, Cousyn L, Soleil D, Fernandez-Vidal S, Mathon B, Houot M, Lemaréchal JD, Clemenceau S, Hasboun D, Adam C, Navarro V (2022) Long-term deep intracerebral microelectrode recordings in patients with drug-resistant epilepsy: Proposed guidelines based on 10-year experience. NeuroImage 254: 119116.

Malach R (2021) Local neuronal relational structures underlying the contents of human conscious experience. Neurosci Conscious 2021: niab028.

Maris, E, Oostenveld, R (2007) Nonparametric statistical testing of EEG- and MEG-data. J Neurosci Meth 164: 177–190.

Mc Kinney W (n.d.) Pandas: A foundational python library for data analysis and statistics. Retrieved October 19, 2023, from https://www.dlr.de/sc/portaldata/15/resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf

Miccoli B, Lopez CM, Goikoetxea E, Putzeys J, Sekeri M, Krylychkina O, Chang SW, Firrincieli A, Andrei A, Reumers V, Braeken D (2019) High-density electrical recording and impedance imaging with a multi-modal CMOS multi-electrode array chip. Front Neurosci 13: 641.

Okada K (2013) Is omega squared less biased? A comparison of three major effect size indices in one-way ANOVA. Behaviormetrika 40: 129–147.

Oostenveld R, Fries P, Maris E, Schoffelen JM (2011) FieldTrip: Open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. Computat Intell Neurosci 2011: 156869.

Quiroga RQ, Reddy L, Kreiman G, Koch C, Fried I (2005) Invariant visual representation by single neurons in the human brain. Nature 435: 1102–1107.

Rainer G, Asaad WF, Miller EK (1998) Selective representation of relevant information by neurons in the primate prefrontal cortex. Nature 393: 577–579.

The pandas development team (2020) pandas-dev/pandas: Pandas 1.0.5. In Zenodo. https://doi.org/10.5281/zenodo.3898987.

Rutishauser U, Schuman EM, Mamelak AN (2006) Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo. J Neurosci Meth 154: 204–224.

Sassenhagen J, Draschkow D (2019) Cluster-based permutation tests of MEG/EEG data do not establish significance of effect latency or location. Psychophysiology 56: e13335.

Sargolini F, Fyhn M, Hafting T, McNaughton BL, Witter MP, Moser MB, Moser EI (2006) Conjunctive representation of position, direction, and velocity in entorhinal cortex. Science 312: 758–762.

Solstad T, Boccara CN, Kropff E, Moser MB, Moser EI (2008) Representation of geometric borders in the entorhinal cortex. Science 322: 1865–1868.

Sternberg S (1966) High-speed scanning in human memory. Science 153: 652–654.

van Vliet M, Liljeström M, Aro S, Salmelin R, Kujala J (2018) Analysis of functional connectivity and oscillatory power using DICS: From raw MEG data to group-level statistics in Python. Front Neurosci 12: 586.

Viejo G, Levenstei, D, Skromne Carrasco S, Mehrotra D, Mahallati S, Vite GR, Denny H, Sjulson L, Battaglia FP, Peyrache A (2023) Pynapple, a toolbox for data analysis in neuroscience. eLife 12. https://doi.org/10.7554/eLife.85786.

Wang Y, Yang X, Zhang X, Wang Y, Pei W (2023) Implantable intracortical microelectrodes: reviewing the present with a focus on the future. Microsystems Nanoengineering 9: 7.

Wiesel TN, Hubel DH (1963) Single-cell responses in striate cortex of kittens deprived of vision in one eye. J Neurophysiol 26: 1003–1017.

Yger P, Spampinato GL, Esposito E, Lefebvre B, Deny S, Gardella C, Stimberg M, Jetter F, Zeck G, Picaud S, Duebel J, Marre O (2018) A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. eLife 7. https://doi.org/10.7554/eLife.34518.