



# UNIVERSITAT<sub>DE</sub> BARCELONA

## Final Degree Project **Biomedical Engineering Degree**

### **“Spike Sorting of Human Single Neuron Signal Recordings”**

Barcelona, 8th June 2022

Author: Àlex Castillo Camacho

Director: Albert Compte

Tutor: Santiago Marco

## **Abstract**

Neuroscience is a field in constant evolution, searching for better ways to understand human brains and behaviour. Challenged by the complexity and difficult accessibility of the brain, neuroscience must constantly be at the edge of scientific and technical developments, learning new ways to obtain and understand neural data.

The approach of this project departs from one such novel findings, human single neuron signal recordings, an emerging technology with a great many applications in research fields but currently limited by the lack of extensive data and previous research. Invasive cell-level recording techniques have been extensively used since its inception in animal brain studies, where they have contributed to wide-ranging research and count with consistent and standardized procedures for its application and analysis. On the other hand, its potential use in theoretical human research is still underdeveloped and does not count with a comprehensive and systematic data analysis framework. However, human single neuron data has the potential to path the way for state-of-the-art research focusing on the behaviour of deep neural processes, such as learning and cognition.

The main objective of the project is to adapt pre-existing single-neuron data-analysis software for animal studies to human recordings, creating a continuous dataflow from the obtention of the data to its ultimate analysis.

## Table of Contents

1.	Introduction .....	4
	Research background .....	4
	Technological background and market analysis .....	6
	Scientific objective and technical question.....	8
2.	Methods .....	10
	Concept engineering .....	10
	Detail engineering.....	14
	Preprocessing and filtering – Matlab .....	15
	Spike sorting – Kilosort.....	17
	Manual refinind – Phy.....	20
	Troubleshooting and alternative steps .....	29
	Troubleshooting and error solving .....	29
	Alternative steps.....	31
3.	Results .....	33
	Pre-processing and filtering .....	33
	Spike sorting.....	36
	Manual refining .....	39
	Subsequent analysis .....	42
4.	Discussion .....	44
	Project objective and results overview .....	44
	Limitations and future development.....	45
	Economic study and budget .....	47
	Confidential aspects .....	48
	Execution chronogram.....	49
	SWOT analysis.....	50
5.	Bibliography .....	52

# 1. Introduction

## Research background

Sleep is a universal human need and the lack of it severely impairs cognition and every aspect of our life. One of the main purposes of sleep is linked to synaptic plasticity and homeostasis. Cortical neuronal activity is significantly influenced by wakefulness and sleep. Sustained wakefulness, represented by sleep pressure, has been shown to increase the firing rate of cortical neurons in all behavioural states (Vyazovskiy et al., 2009). On the other hand, sleep and thus low sleep pressure induces decreased firing rate of cortical neurons, as a function of sleep homeostasis. This dependence of firing rate on sleep pressure is observed systematically in all behavioural states, from wakefulness to REM sleep to NREM sleep (Vyazovskiy et al., 2009).

Slow wave activity (EEG power between 0.5 and 0.4 Hz) is a main indicator of sleep depth, and it plays a key role in sleep regulation and sleep homeostasis (Riedner et al., 2007). Synaptic homeostasis has been proposed to oppose synaptic strengthening produced by wakefulness, thus allowing for synaptic plasticity. Increased synaptic strength is highly energy consuming and requires great cellular supplies, causing cellular stress (Tononi & Cirelli, 2014). For this reason, a continued strengthening of synapses leads to saturation of the ability to learn. During sleep, synaptic plasticity is enabled by renormalizing synaptic strength (Tononi & Cirelli, 2014). More importantly, synaptic plasticity is thought to be a key mechanism in memory processing, learning and cognition (Hughes et al., 2010). A great number of studies have linked short to long periods of sleep with processing and retention of declarative and procedural memory, with little to no evidence of reducing memory formation. Sleep has a function in the consolidation of emotional information as well (Diekelmann & Born, 2010).

Two main hypotheses for mechanisms explaining synaptic plasticity have been formulated. On the one hand, the active system consolidation hypothesis suggests that during slow wave sleep (SWS) there is a repeated re-activation of the encoded memories, which leads to the affected memories being accentuated. On the other hand, the synaptic homeostasis hypothesis prompts the idea that during wakefulness a net increase in synaptic activity takes place, which needs to be downscaled during the night in order to sustainably provide the brain with synapses to reuse for the following waking period. Thus, the reduction in amplitude of SWS during the night would result in selective strengthening of already significant synapses, while nullifying weaker ones, obtaining a higher signal-to-noise ratio in reinforced memories (Kanthida van Welzen).

SWS play a significant role in both hypotheses and are therefore considered a key element of sleep and synaptic plasticity. It has been formulated that SWS respond proportionally to periods of sleep and wakefulness, having larger intensities in sleep sessions occurring after longer waking times, which enact lower sleep pressure, with respect to shorter prior waking times. On the same lines, prolonged sleep periods after sleep deprivation leads to a continuous reduction of SWS intensity. In cases of SWS sleep disruption, following periods show an increased initial strength of SWS, indicating a lack of intensity reduction in the previous session (Borbély et al., 2016).

The processes that control and regulate synaptic plasticity are widely unknown. However, among several hypotheses, some studies have linked overnight slow wave slope with anti-N-methyl-D-aspartate receptor (NMDAR) (Lau & Zukin, 2007; Shepherd & Huganir, 2007). Central to neurological functions is synaptic dependence in excitatory NMDA and AMPA receptors. In humans, the effects of the presence or lack of these receptors have been demonstrated to impact memory, learning, cognition, and psychosis from indirect approaches, like pharmacological trials (Gunduz-Bruce, 2009). More directly, it has been studied the effect of anti-NMDA receptor encephalitis, in which antibodies against NR1-NR2 heteromers of the NMDA receptor are targeted, producing a characteristic neuropsychiatric syndrome, including behavioural symptoms, rapid memory loss, seizures, abnormal movements, hypoventilation and autonomic instability (Dalmau et al., 2008). Interestingly, many symptoms have been observed to be reversible, receding when the concentration of NMDAR increases again due to treatment of the disease (Ishiura et al., 2008). Overall, symptoms grow in severity with the increased removal of surface NMDA receptors, which indicates a relation between NMDAR, and cognitive stability and preservation (Hughes et al., 2010).

In a 2007 study (Lau & Zukin, 2007), the proposed model of reduction of SWS during sleep, which is linked to memory consolidation and learning, was tested using EEG recordings from humans. NREM sleep from the beginning of the night and NREM sleep from the end of the night were compared, with every other parameter normalized and no significant disturbance in sleep or the recording. SWS were identified measuring the slope waveforms of the EEG as a marker for synaptic activity. As expected, SWS signals showed a marked decline from early to late night in spectral power analysis, which is represented in Figure 1. The maximum slope measurements were mainly in the first part of the night, with median declines of approximately 19% across all slope measurements. Early sleep SWS was associated with higher-amplitude waves and in larger proportions.

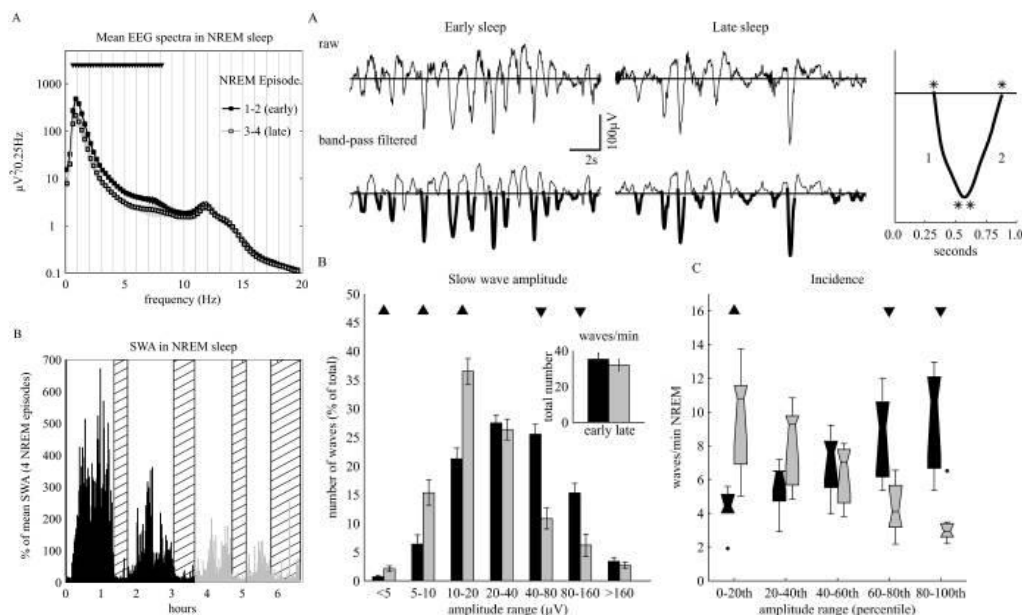


Figure 1. Left figure: mean EEG power spectra in NREM sleep at the beginning and the end of the night. Right figure: measuring method for spike amplitude and statistics. (Riedner et al., 2007)

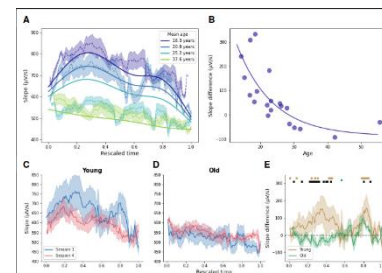
In a more recent study (Kanthida van Welzen), a similar approach has been considered for studying the evolution of SWS during the night. However, this time EEG data from humans was not considered only during early- and late-night periods but was taken as a continuous parameter to quantify SWS intensity. Additionally, control patients have been compared with schizophrenia and anti-NMDAR encephalitis, to analyse the effect of synaptic plasticity blockage due to NMDAR reduction in illness with SWS intensity reduction. Similar to the other study, wave amplitude was measured using slope criteria and pre-processed to remove noise. Then, instead of choosing fragments from specific parts of the night, all NREM 3 stage fragments were analysed, preferring N3 over N2 stage due to the main presence of slow waves in N3. Finally, all SWS were represented continuously, to observe SWS amplitude progression during the entirety of the night.

The results showed a clear slope decrease overnight, but slope change pattern was significantly dependant on age. As opposed to previous models and studies, younger participants had a marked increase in slope at the beginning of the night, with later diminution leading to homeostatic reduction of synapsis strength. Such pattern was less prominent with age and was only present until around 30 years old, as can be observed in Figure 2. Moreover, comparing control results of schizophrenia and anti-NMDAR encephalitis patients, the latter demonstrated less step declines in SWS slopes, resulting in lower reductions of slow waves during the night.

The slow intensity results compared with age, which was a not-intended outcome of the research, fits earlier studies using discreet approaches, such as described above. In both cases, difference in SWS amplitude from the beginning to the end of the night had similar results for all ages. However, in younger participants this new study found that slow waves periods showed a marked increase in intensity during the beginning of the night, following wakefulness, which then led to a steeper reduction in general amplitude towards the end of the night. These findings support both hypotheses of synaptic plasticity in SWS. The synaptic homeostasis hypothesis is reinforced by the confirmation of synapse renormalization leading to reduction in amplitude, while the early night potentiation observed in younger participants indicate a reactivation of synaptic paths to encode and consolidate memories, supporting the active system consolidation hypothesis. However, the presence of such reactivation solely in children and young adults could also indicate a developmental need of larger synaptic plasticity transformations, reflected in structural as well as functional changes.

## Technological background and market analysis

Non-invasive brain recording techniques are limited by the general scope of the methods and the lack of specificity. In order to study detailed processes in the brain, invasive techniques with better spatial and time resolution are needed. In this field, it is of special interest the use of human single-neuron recordings, which provides direct information at the cellular and intercell level. Single-neuron data is decisive to develop our scientific hypothesis concerning brain plasticity due to the need of cell-level firing recordings to study cell connectivity (Kubska & Kamiński, 2021).



**Figure 2. Influence of age on overnight slope change.** A. Dashed lines: the overnight slope  $\pm$  SEM for control subjects split into quartiles. Every quartile contains five control subjects, shown is their mean age. Solid line: the overnight slope as predicted by the age-controlled slope model. The age-controlled slope model looks into overnight time, age and their interaction with knots at  $t=0.25, 0.60, 0.80$ . B. The slope difference in early night ( $t=0.00-0.25$ ) per age with exponential fit. C-D. Overnight slope  $\pm$  SEM in session 1 and 4, one year after, for young (C) and old control subjects (D). Twelve control subjects underwent both sessions and were split up by the median into a young and old group. E. The slope difference between session 1 and 4 for young and old control subjects. The yellow and green solid squares in the upper part of the figure mark significant differences between session 1 and 4 for young and old participants, respectively (one-sided paired permutation test:  $p < 0.05$ ,  $n = 6$  participants). The black solid squares mark significant differences between young and old participants (one-sided permutation test:  $p < 0.05$ ,  $n = 12$  participants).

Figure 2. Top figures: slow wave sleep intensity as a function of time during the night. Bottom figures: SWS slope overnight depending on age. (Kanthida van Welzen),

The main limitation of invasive brain recording is the obligation to implant needle-like electrodes into the human brain, which cannot be pursued due to medical and ethical reasons. The most common situation nowadays in which such research can be undertaken is in cases of pharmacologically intractable epilepsy, where the introduction of electrodes into the brain is justified on the grounds of health monitoring and preparation for surgery (localization of focus for seizure onset) (Quiroga et al., 2005). In such cases the so-called “Behnke-Fried” electrodes are used, which were developed in the 1990s and consist of depth electrodes with microwires protruding from the end. Each electrode has eight microwires at the extreme, which obtain direct recordings from single neurons (Kubska & Kamiński, 2021).

The design of depth electrodes is limited by its application in humans and thus has several constraints. The main one is the characteristics of the microwires, which by reason of their small dimensions and the problematic environment in which they are inserted, do not have a fixed shape but, as wires usually behave, can bend and change position. This matter adds an additional level of difficulty to the analysis of single neuron data, since position is usually unknown and can shift over time. Another concern about microwire depth electrodes is the lack of specific methodological procedures for successful recording implementations and troubleshooting, which has led to the technology being restricted to a few clinical sites worldwide and consequently to limited availability of single neuron data (Misra et al., 2014).

Despite still being an emergent technology in humans, spike detection in single neuron recordings has been used in rat and mouse research for a longer period. Lower ethical constraints allow for the implantation of more complex electrode devices and distributions, which support large scale recordings with high numbers of channels. To properly interpret this data and automatize the spike sorting process, a scalable and accurate software is required, which minimises time spent in manually analysis raw data. Among such applications, KiloSort was developed in 2016 to “implement an integrated template matching framework for detecting and clustering spikes from multi-channel electrophysiological recordings” (quote) (Kilosort Documentation). KiloSort is mainly aimed at mouse recordings or similar research, due to its development around invasive single-neuron-recording devices. It boasts an intuitive and straightforward graphical user interface and a series of parameters that can be modified in order to increase accuracy and adaptability of the spike detection and sorting.

Kilosort consist of a template-matching algorithm which detects neuron spikes in different channels from brain activity recordings. One of the main applications is identification and grouping of neuron-firing patterns to study how such neurons inter-behave and evolve over time. Kilosort is optimally used together with Phy, an open-source manual clustering Python library with a graphical interface designed to improve manual refining of automatic spike sorting (Phy Documentation). Kilosort output data is arranged to be directly collected by Phy, with the results files of Kilosort containing all information needed to implement the manual interface of Phy.

Both are developed by Cortextlab, a GitHub collection of repositories with several popular scientific software for the visualization and analysis of brain signals. Cortextlab is powered by the University College of London (UCL) and combines experiment and computational analysis to understand sensory perception and decision-making mechanisms in the brain. Apart from Kilosort, Phy also

supports data sorted with Spyking Circus, klusta and klustakwik2. The current version of Phy (2.0 beta) was first released on February 2020 (Phy Documentation).

As opposed to Kilosort, Phy does not allow for automatic analysis of data, but rather offers a set of visualization and statistical tools to understand the results of spike sorting and try to manually refine the automatic algorithm, handling exception or anomalies and adding a human layer of expertise to the process. As such, Phy goes hand in hand with Kilosort and both are usually considered together, even if they could be used separately and even paired with external software. The general idea is to let the automatic algorithm do the time-consuming and massive tasks, while investing expert human resources on the improvement of output data in an optimized and advantageous environment.

Moreover, regarding physiological signal recordings, many programming languages have specific packages available to study, process and analyse such data. Among these, MATLAB® is a matrix-based programming language and built-in programming graphics environment, with extensive tools to work with computational mathematics and many types of data (Matlab Documentation). Its effective combination of tools and the graphical interface have made Matlab increasingly popular among data scientists and boasts a resourceful and highly technical user community. Mainly operating in desktop applications, it can also be combined with other programming languages in online as well as offline settings. Matlab was created by MathWorks, a company specialized in the development of mathematical computing software for researchers, scientists, engineers and mathematicians. Apart from Matlab, MathWorks also produces Simulink, a graphical environment for simulation and model design systems. The purchase of MathWorks products is based on licenses, which can be perpetual or annually acquired (Matlab Documentation).

One of the most enticing aspects of Matlab is the wide availability of additional toolboxes, called Add-Ons, which provide additional functionality for specific tasks and applications, extending the capabilities of the system. Packages can be obtained directly from the Add-On Explorer of Matlab or added manually from external providers. Their usage is independent of Matlab and as such can be free of charge or requiring additional purchases (Matlab Documentation).

### **Scientific objective and technical question**

Following aforementioned research on synaptic plasticity as a function of SWS intensity during the night, this project developed from an initial scientific objective. As recent studies have shown, synaptic plasticity is highly dependent on slow wave activity, even if the source of this relation is not completely well-known, and SWS has been observed to decrease during the night. Such decrease in intensity is theorized to foster memory consolidation by enhancing memory pathways encoded in synaptic connections, while nullifying less predominant memories, thus obtaining both memory preservation and synaptic homeostasis by the end of a regular extended sleep period. However, most of these studies have been developed using whole brain imaging techniques, such as EEG, and thus offer statistical interpretations to explain SWS behavioural changes in specific brain regions.

With the development of human single neuron recording techniques, researchers now have access to data from singular neurons, allowing the study of specific synaptic parameters in a direct way,



instead of through statistical inference. Taking advantage of this relatively new technology, the initial scientific objective for this project aimed to characterise synaptic plasticity changes during the night using neuron connectivity data instead of whole-region information. To do so, the synaptic homeostasis hypothesis was considered: we wanted to analyse whether SWS intensity decrease, exposed by multiple studies, was effectively related to selective activation of specific neuron pathways, thus leading to memory fixation. Hence, we hypothesized that neuron connections during the night would evolve differently according to whether each neuron synapsis was reinforced or nullified.

In order to support or invalidate our hypothesis, we needed to obtain single neuron data from a whole night of sleep using micro-wire electrodes, extract all firings present in the recording and sort them into specific neurons, group the obtained neurons into synaptic pathways using correlation mechanisms, and analyse how each pattern evolved through the night. Ideally, data obtained from NREM 3 stage fragments of the night (since it the stage containing most slow wave activity) would contain pairs or groups of connected neurons, and the firing rate of such groups could be interpreted to represent synaptic strength during the sleep period.

However, the technical realization of this objective incurs in several difficulties. Firstly, the structural instability of human single neuron recording devices can lead to unstable data, in which the position of electrodes shifts at different points during the night, thus removing the continuity attribute of detected neurons necessary for the progression analysis. Moreover, considering the small number electrodes of which we disposed, it would be highly improbable to find signals obtained from different specific neurons, and even more so to be able to correlate enough of them with the purpose of obtaining experimentally significant results.

However, the most prevalent limitation is the scarce or lack of software, documentation and research of spike detection and sorting of human single neuron recordings. Being a novel and still-developing technology, little research about the data that can be obtained or how to process it is available. Additionally, any software capable of spike detection and sorting using single neuron data has been developed for and specialized in animal recordings, without tools to adapt it to the characteristics of human recordings devices or the features of human signals. Such limitations were the source for the technical question which this project is based on: the development of a complete pipeline for the extraction, detection and sorting of action potential spikes from single neuron recordings. The objective was to ideate and implement a series of methods which, employing priorly-developed high-input techniques from other fields (such as animal single neuron recordings), would provide an ordered and consistent pipeline for the management of data from raw recordings to neuron-sorted spikes information, which can be directly applied in scientific research.

## 2. Methods

### Concept engineering

The processing software is divided in three main methods: preprocessing and filtering, spike sorting, and manual refining. Each of these is developed in different applications, which may require separate environments.

Preprocessing and filtering are carried out in Matlab, using specialized toolboxes. Matlab is specialized in operating with whole matrices and arrays, so basic methods include creating variables, array indexing, arithmetic, and data types (Matlab Documentation). Matlab releases new versions of the software annually, including bug fixing; Matlab R2019a was used for this part of the project. A Matlab script has been developed, which comprises all the necessary methods to read raw data, pre-process the signal, filter noise and prepare the output data for the spike sorting procedure. Matlab offers a number of Add-Ons which can extend the functionalities of the basic interface, presented in Figure 3, allowing for the usage of many Toolboxes at the same time.

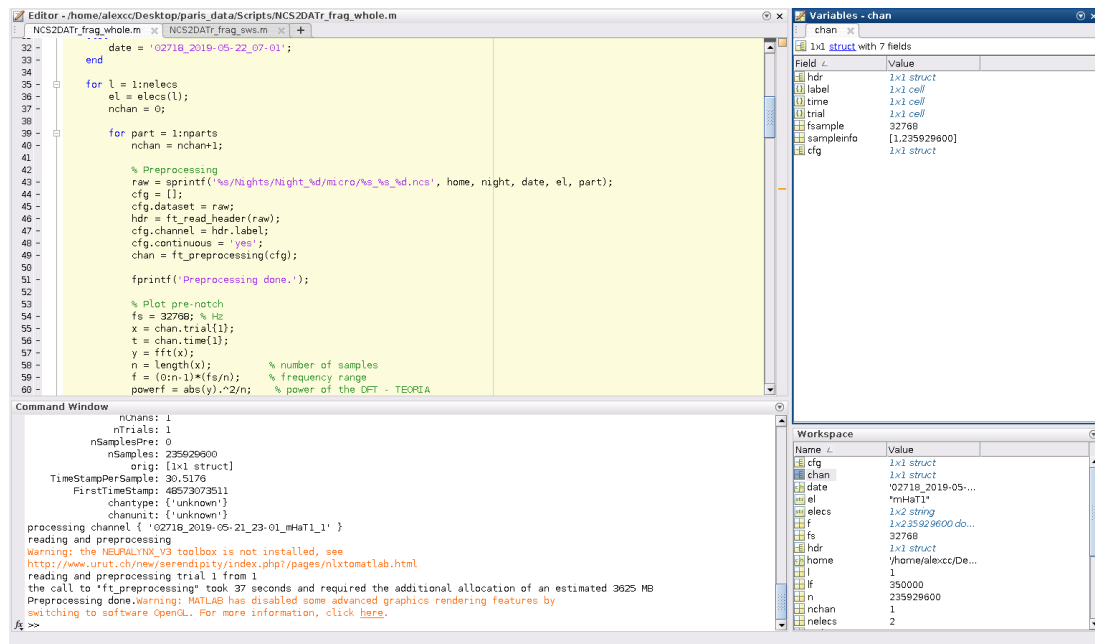
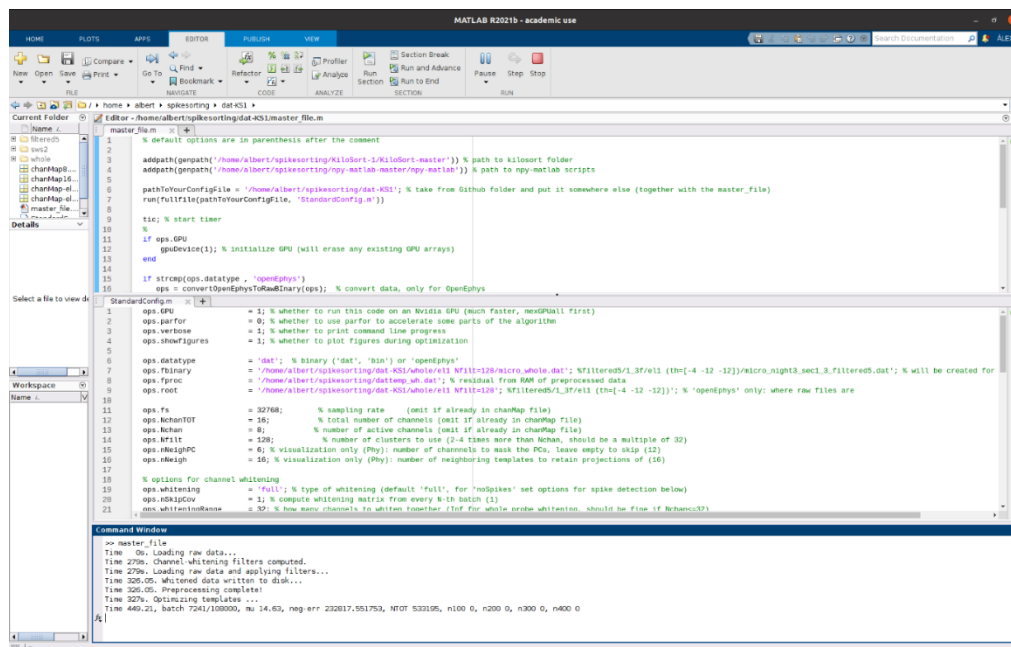


Figure 3. Example of Matlab interface while executing the pre-processing code. The interface is divided in several windows, which can be distributed freely. The four windows shown in this Figure, in from top to bottom and from left to right are: Editor, Command Window, Variables, Workspace.

For this project, two toolboxes are employed: Fieldtrip and DSP System Toolbox. Fieldtrip is an open-source free of charge software released under the GNU general public license. It is dedicated to MEG, EEG and other electrophysiological data analysis, using Matlab implementation. It includes algorithms for every level of analysis and allows the adequate manipulation of data with high efficiency and preventing information loss. Mainly Fieldtrip was used to read the raw data, structure it properly, and obtain spike-sorting-oriented files. DSP System Toolbox, a licensed toolbox integrated in Matlab, provides algorithms and visualization tools for signal analysis. It contains a wide variety of functions, but for the purpose of our data the main interest is the filtering methods (DSP System Toolbox Documentation).

Regarding spike sorting, as mentioned before Kilosort is an automatic spike sorting software developed in 2016 by Cortextlab, that uses a template matching framework to identify and group neuron signals from raw single neuron recordings. Kilosort is specifically designed to process data from mouse and rat recordings, due to the well-established characteristics of such research and the large number of signals they usually deal with. However, the same algorithm can be applied to any kind of single neuron data. In this project, the parameters of the analysis are adapted to respond optimally to human single neuron data, which tends to be significantly smaller-scale and of lower quality due to the characteristics of the electrodes.

Four versions of Kilosort have been released since its initial development, mainly focused on improving drifting clusters' tracking and the creation of a graphical user interface (GUI) from Kilosort 2.0 onward, which can be seen in Figure 4. However, some improvements can prove detrimental in the analysis of specific recording configurations, such as tetrodes or single-channel recordings, and thus empirical testing of a specific version to assess any possible shortcomings is recommended (Kilosort Documentation).



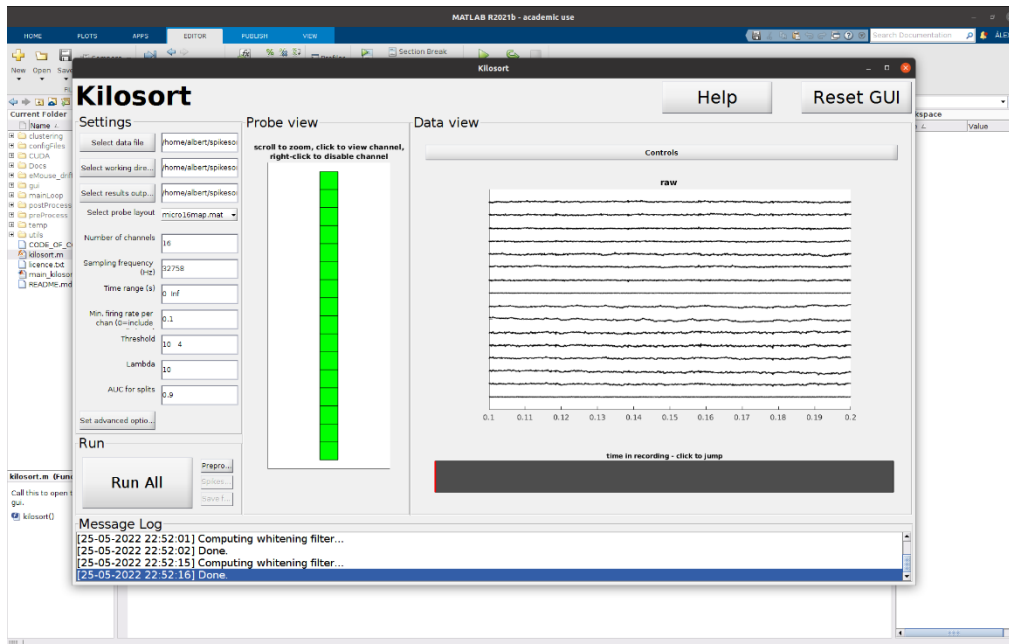


Figure 4. Top figure: Kilosort 1 interface. Bottom figure: Kilosort 2.0 and onward GUI (bottom). Both interfaces of Kilosort are captured after charging a 16-channel DAT file.

After thorough investigation of the methods available and possible prior experience from other researchers in the human single neuron field (with very limited results), the preference of KiloSort 1 over more updated versions of the software was decided based on the characteristics of our recordings and direct recommendation from source developers, considering only Kilosort 1 and 2.0 suitable for our data. The main limitation was the lack of a defined structure and geometrical distribution of microwires, since they can freely bend and move. Hence, a map of channel allocation cannot be confidently provided and improvements heavily relying on channel map would be severely affected.

Data can be manipulated in Kilosort either by modifying the source code, developed entirely on Matlab, or using the graphical user interface (GUI). The GUI is intended as a launcher for Kilosort and is only available from version 2.0 onward. Consequently, working with Kilosort 1, the source files were directly accessed in order to analyse the data. Kilosort offers a number of parameters related to spike detection and sorting which can be modified in the files to alter the software's behaviour. We mainly focused on the adjustment of channel selection attributes and sensitivity of neuron identification.

Lastly, in conjunction with Kilosort, the spike sorting is completed by Phy. Also developed by Cortexplab, Phy does not involve any kind of automatic analysis as the two previous steps but is actually meant to be used as a manual refining tool. To do so, it provides a graphical user interface containing a wide variety of data visualization tools and representations of statistic magnitudes. In this way, the user is able to detect problems or ambiguities in the automatic sorting, as well as to correct them performing merges and splits of spike clusters, without any kind of algorithm supervision. Additionally, neuron clusters containing all spikes sorted together can be classified and labelled for later study. Neuron clusters are usually groupings of spikes generated by the same

neuron, but there can also be instances of MUA or noise, which are the main labels available in Phy.

The graphical user interface of Phy, as can be observed in Figure 5, is divided into several windows which can be selected independently. Each window contains a calculation or data plot meant to address one aspect of spike sorting, and also incorporates a series of parameters which can be modified to obtain more accurate or diverse visualizations. The main three types of windows available in Phy, divided according to their function and source, can be classified as:

- Spike sorting results and cluster manipulation: windows aiming to represent the clustering results of the spike sorting, most importantly the ClusterView with a list of all present clusters, and tools for directly comparing between clusters.
- Data visualisation: direct representation of the raw data or specific characteristics of it, such as the waveform, the amplitude or the spike position in time.
- Calculations: another key aspect of the software, it performs statistical and arithmetic calculations to better compare different clusters and understand how they fit into one category or the other. Some examples of calculations can be the auto- and cross-correlograms or the feature comparison, based on the principal components analysis.

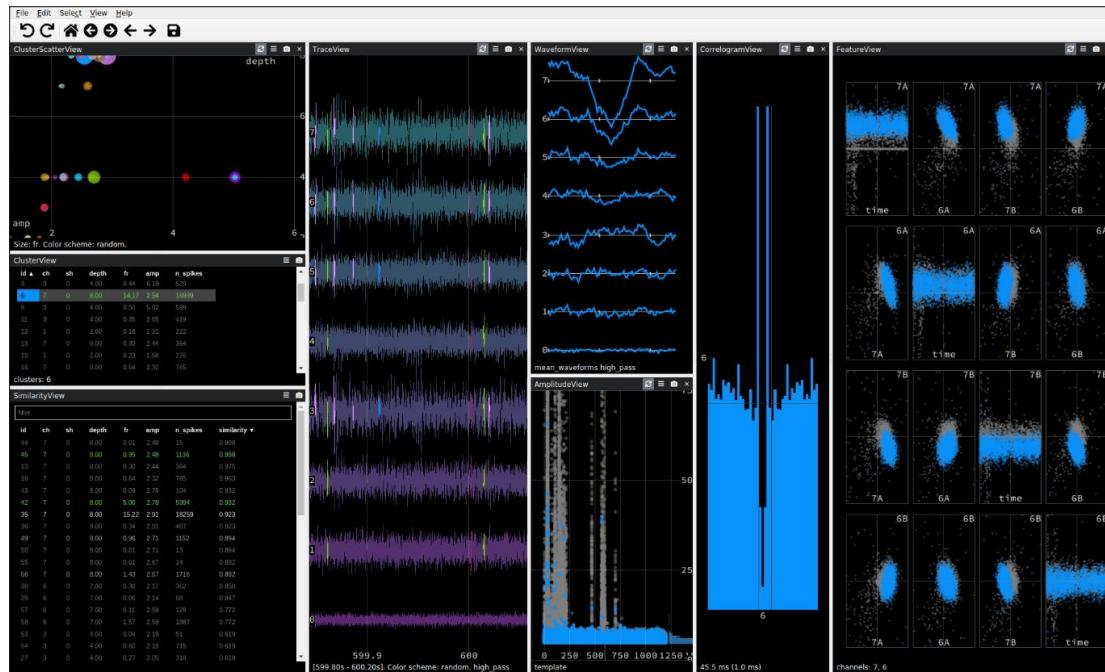


Figure 5. Graphical user interface of Phy.

On top of the available predetermined windows, Phy offers the possibility to add plugins to the software, extending the GUI or modifying certain aspects. Plugins are written by users in Python programming language following pre-set templates and must be integrated into the Phy configuration files and activated. Plugins are an easy and accessible ways to customize the interface and add more functionality to the software. Plugins can access virtually any element of Phy, so they can significantly improve the versatility of its features and enable new actions to be performed, such as emitting events, extracting waveforms, creating new views, adding the number of spikes in views, defining a custom cluster metrics and so on.

In order to curate the automatic spike sorting executed by Kilosort, Phy allows the user to perform two basic actions: merging/splitting and labelling. Merging and splitting are Phy operations which allow to respectively combine two clusters or divide a cluster into two, in order to unify spikes from the same neuron or separate spikes from distinct neurons. These actions are performed manually on any of the views that represent spikes temporally, such as the AmplitudeView or the FeatureView (in depth explanation of each view in the Detail engineering), by selecting groups of spikes and merging or splitting them. Labelling, on the other hand, lets the user classify each cluster by its contents, inferred by the user from observation of parameters in Phy and their own experience. Those labels can be used to discard noise clusters and for post-analysis research. By default, Phy offers four classification labels:

- “good”: for clusters containing spikes identified as produced by the same single neuron.
- “mua”: for clusters with spikes belonging to different neurons which cannot be split due to lack of differentiating features.
- “noise”: for clusters containing exclusively or mostly noise.
- “unsorted”: for other types of clusters which cannot be classified or are deemed unfitting for any of the prior categories, the user can choose to leave the unsorted.

## Detail engineering

As described before, the software is divided into three methods, which need to be run independently and are directly connected, that is each output is readily prepared to be imported into the next step of the procedure. Each method consists of a pipeline of actions, comprising code execution, parameter definition and interactive visualization, which are optimized to simplify and enhance the analysis of human single neuron data.

The three methods are described together with the software application that enables its performance:



The input data that goes initially into the preprocessing step are CSC files, terminating in *.ncs*. CSC files are produced by any Neuralynx acquisition software, like Cheetah or Pegasus, and encode Continuously Sampled Channels (Marius Pachitariu et al., 2016). Each file is divided into two sections: the File Header, which contains the system settings and data attributes of the signal acquisition, and the File Data, stored in binary format. The output data are a series of files containing spectral, temporal and statistical information about the spikes found, as well as information about the sorting.

## Preprocessing and filtering – Matlab

Data is imported into Matlab using the Fieldtrip toolbox, which encompasses a series of functions designed to open electrophysiological signal files, among which is CSC. All information about the data, such as sampling frequency and length of the data, is included in the file's header. The function "ft\_preprocessing" imports the data as a Matlab struct. The importing, preprocessing and filtering methods must be applied independently to each file, so that the whole procedure is iterated for every set of data available using Matlab loops.

One limitation of Matlab is the availability of memory, which defines the maximum size of data that can be worked with. This constraint can lead to memory errors when working with large datasets and scripts that create many substantial variables. Additionally, this error can only be solved either by maximising variable-definition efficiency or by increasing the system's available memory. In case of adapting the hardware to dispose of more memory, some parameters need to be changed in Matlab as well to put the adaptation into effect: Matlab array size limit, which defines the maximum size that an array of data can be, must be increased, as well as the Java heap size limit, which determines how much memory is available for Java objects. However, signal recordings can be very long, so for memory reasons or any other, fragmentation can be done using Fieldtrip. Similar functions, like resampling or redefining timestamps can also be obtained from Fieldtrip but were unnecessary for this data, especially considering the reduced physical and temporal dimensions of neuron firings, which require high sampling frequencies to have enough resolution to conduct the spike sorting and thus a down-sampling would be detrimental.

Once all preprocessing functions have been applied, before structuring the data for spike sorting, it must be filtered. Single neuron recordings have been observed to incur a significant number of artifacts and noise signals, which need to be removed for efficient post-processing. Artifacts can be identified using whole-brain imaging techniques, or as in the case of this project can be dealt with in later steps, since they are mostly discarded during the automatic spike sorting. External noise signals on the other hand, occurring at specific frequencies, can be most efficiently removed using a notch filter. The function employed is "iirnotch()" from the DSP System Toolbox, which implements a second-order IIR notch filter. The inputs of the function are the notch frequency ( $W_0$ ) and the 3 dB bandwidth (BW).

Before implementing the filter, a spectral analysis of the data is performed to identify potential noise-derived frequency alterations, represented as a function of their power spectrum calculated via discrete Fourier transform. Noise has been considered to be especially significant at lower frequencies, so filtering is limited to the range between 2.7304 and 273.0665 Hz. This range has been defined experimentally, observing the first noise signals at around 30Hz (with generally a significant peak at 50Hz) and still substantial noise frequencies up to 250Hz and 266Hz approximately. As such, this range can be modified depending on the data and the characteristics of the signal.

Considering the high resolution of single neuron data, the manual identification of noise frequencies has been deemed too imprecise, hence relying on the development of an iterative algorithm which automatically detects and implements the notch filter. All the parameters of this algorithm are variable and calculated with statistical features of the signal, so that both peak identification and

power reduction due to the filter are tailored to the characteristics of the frequency and the overall recording. The  $W_o$  parameter refers to the frequency that is required to be removed and is calculated as the notch frequency divided by the Nyquist frequency, which is equivalent to half the sampling frequency of the signal:

$$W_o = \frac{f_{notch}}{f_{sampling}/2}$$

The  $BW$  parameter, on the other hand, concerns the amplitude reduction power of the filter, and is calculated as  $W_o$  divided by  $Q$ , the quality factor of the filter:

$$BW = \frac{W_o}{Q}$$

Modifying  $Q$ , the intensity of the filter can be adjusted, in order to remove only the necessary amplitude of the notch frequency without altering the surrounding frequencies. It should be noted that an increase in  $Q$  leads to a decrease in  $BW$ , which diminishes the attenuation of the filter, and vice versa. After a process of trial and error observing discrepancies in several different signals, the preferred algorithm to calculate the  $Q$  value was defined as:

$$Q = \frac{f_{notch} * 35 * \text{average power}}{\text{power of the } f_{samp} - \text{median power}}$$

In the presented formula,  $Q$  depends on the notch frequency and the difference between the power of the notch frequency and the median power of the signal in a window of  $\pm 3000$  samples the notch frequency, multiplied by 35 times the average power of the signal in the same  $\pm 3000$  samples window. The “35” constant of proportionality was decided on experimentally as the most suitable value for all the data evaluated, optimizing the trade-off between acceptable notch frequency removal and no excessive alteration of the baseline data. The objective of this algorithm was to be as general as possible, so that the same expression could be applied to all the data. However, the constant value can be used to tune the effects of the filter and should be modified in every defined set of signal recordings on which the software is implemented. Then the filter is implemented using the “filter” function of Matlab. The output signal, examined with a post-filtering spectral analysis, presents significantly or completely decreased power of noise signals, without alterations of the non-filtered frequencies.

Afterwards, the data is structured according to the characteristics required by the spike sorting software. In case of having several electrodes or channels working parallelly in the same time frame and same sampling frequency, there are grouped and merged together in the same array with a common time series, using the Fieldtrip function “ft\_appenddata”. This way, all metadata and recording information about the signals is kept. On the other hand, for multiple fragments of data belonging to different time parts of the same recording, hence with equal sampling frequency, scale and preprocessing but different time stamps, they must be manually appended in the correct order, also in a single array with all necessary information. Additionally, an initial iterative loop has been developed to help extract exact fragments of the data in case of needing to process only a part of the recording (such as in the case of sleep periods during the night). This piece of code is meant



to help users find positions in the data using time stamps and is integrated in the main loops of the script so that data defined in this way is processed correctly.

Ultimately, the resulting array with data from different channels and fragments must be stored in DAT format, a generic data file typically containing data in binary or text format. This is the file format required by Kilosort, however in case of another spike sorting software being used, it can be modified to adapt to its individual requirements. For the case of Kilosort, the data must be saved specifically in integer 16 format (signed 16-bit integer). For research purposes, all plots obtained during the processing of the data, namely power spectrum of the data prior to and after the filtering, have also been saved. The chosen formats have been FIG image format and PDF document.

### **Spike sorting – Kilosort**

Kilosort installation requires the compliance with precise steps to ensure all data is adequately processed and stored. Firstly, Matlab can be downloaded as a ZIP file or directly installed from the Git repository at <https://github.com/cortex-lab/KiloSort>. Then, the “master\_file.m” present in the folder should be copied to the local directory of the data and changed for each experiment carried out, in order to keep any modification made during each. We also added to the local directory the “StandardConfig.m” file, because this is the Matlab script containing all the parameters used in spike sorting, and hence the one that will be modified for each signal analysed. Moreover, for the writing of Kilosort output files, “numpy-matlab” must be downloaded in the working directory. This code allows Kilosort, running in Matlab, to read and write NumPy’s NPY format (“.npy” files), which is native of Python programming language (Kilosort Documentation).

As explained above, due to the high computational requirements of Kilosort, a GPU is certainly recommended, as it will run substantially faster than on a CPU. GPU stands for Graphical Processing Unit and it provides unprecedented computing power by exploiting extreme parallelism which can achieve very high speeds. GPU needs to be implemented in the system and is mainly realized in CUDA, a specialized parallel computing application programming interface which has established itself as the de-facto GPU programming standard for over a decade. CUDA was developed by NVIDIA, a corporation founded in 1993 as a graphics chips company with an outstanding history as a computing units and components developer, being especially noteworthy the invention of the GPU in 1999. To employ the GPU in Matlab, CUDA must be installed, considering which version is more suitable depending on the version of Matlab operating (Oostenveld et al., 2011). Then, mex compilation of CUDA files needs to be set up by executing “mexGPUall” in the Matlab console. As a result, a “mex\_CUDA\_glnxa64.xml” file or similar will have been created in a Matlab root folder along the lines of “matlabroot/toolbox/distcomp/gpu/extern/src/mex/”, which must be copied to the Kilosort folder. Kilosort source code already includes a similar file but that is not compatible with the local environment unless internal paths are modified, so it is important to copy the CUDA file in the corresponding path.

Once installation is complete, the “master\_file.m” file located in our local folder with the data must be run to execute Kilosort. However, in order to run Kilosort all the necessary parameters must be specified beforehand in the “StandardConfig.m” file, including the source paths of the data to

analyse and the features of the spike sorting. Additionally, a channel map file must be created to introduce necessary data distribution information in Kilosort. Input data must contain at least 16 channels for the same time period and be coded in binary integer 16 in a DAT file. In case of recordings with less channels, a proposed solution to the minimum 16 channels problem is to create duplicates of channels or to generate simulated new channels. Later on, in the Kilosort configuration, these additional channels can be opted out of the analysis, thus considering only the correct channels.

Kilosort source code includes a Matlab script (“createChannelMapFile.m”) to specify the characteristics of the data and generate a format-appropriate channel map file. The information about the channels introduced in this file is:

- Number of channels: total number of channels present in the recording, including mock or duplicated channels. Row list of the number of each channel, from 1 to the total number of channels.
- Index of the channels: calculated automatically from the number of channels. Same distribution, row list of serial numbers, but starting from 0 instead of 1.
- Connected: channels that will be considered for the analysis. This variable defined which channels are going to be taken in the spike sorting, so in case of containing mock or duplicate channels, they must be left out in this step. The variable is introduced as a column list of the length of the total number of channels, where active channels are marked True (represented as 1 in Matlab) while not-to-be-considered channels are marked as False (represented as 0).
- Sampling frequency: sampling frequency of the signal, it is the same for all channels.
- Kcoords: it indicates relation between channels, basically for the case of electrodes containing several microwires, in which case each group of channels belonging to the same electrode will have assigned a different number (for example, for the case of having two electrodes with eight recordings each, this variable will be a list of eight instances of 1 followed by eight instances of 2). The information is introduced in a column list of numbers, which will be repeated for channels in the same electrode. This variable is designed to help Kilosort separate electrodes and adequately sort neurons only among close-positioned channels, since long-distance recordings cannot physically perceive the same neuron. However, experimental observation has shown that Kilosort is capable of sorting together spikes from channels belonging to different electrodes even when specified as two different electrodes in Kcoords, so that distance between channels seems to be more dominant than Kcoords in determining whether two spikes can belong to the same neuron in Kilosort. In case of Kilosort mixing independent electrodes in the analysis, it would be convenient to create different channel maps for each electrode where only the channels contained in this electrode are active (indicated in the Connected parameter) and running Kilosort separately for each electrode, using its own channel map. For such procedure to be effective, a different folder for each electrode must be created, containing in every case the same original data and the respective channel map.
- Xcoords: it refers to the localization of each channel in the X axis, and along with Ycoords defines the geometrical distribution of the signals. It is expressed as a column list of the

channel deviations from a 0-centered vertical axis. They are the most important measure in the channel map and play a significant role in spike sorting and the posterior analysis. However, in human single neuron recordings the position of microwires is unknown, so these variables cannot be established. There is no solution for this limitation, and the recommended course of action is to define a simple distribution of channels avoiding any special conformation that could misguide Kilosort, generally on the lines of linear or square organization. That way, no additional information is obtained from the channel map, as opposed to mouse recordings where electrode information is key to study the behaviour of neurons, but at the same time the scope of this inaccuracy is limited (and further reduced by the choice of Kilosort 1 over other versions which attribute more weight to channel location).

- Ycoords: analogous to Xcoords, it defines the position of each channel in the Y axis. It is expressed as a column list of the channel deviations from a 0-centered horizontal axis. It is also unknown and must be inferred as a low-impact distribution analysis-wise.

Every time Kilosort is executed (running the “master\_file.m” file), the configuration parameters in the “StandardConfig.m” file must be indicated appropriately. The Kilosort configuration contains many customization parameters with generic default values which tend to be convenient in most analyses, so the focus of this step is to describe parameters that need to be changed to optimize the spike sorting process. Firstly, all paths containing necessary files have to be adequately introduced in “ops.fbinary”, “ops.fproc”, “ops.root” and “ops.chanMap”: the path of the DAT file to analyse, the path of the general working directory, the path of the folder containing the DAT file to analyse and the path of the channel map design specifically for the DAT file to analyse, respectively. The channel map can be stored in any location, it is not necessary to be kept on the working directory, although recommended. For the case of the “ops.root”, the selected folder will contain the results of Kilosort, a series of output files containing the sorting. Considering these results will be later on supplied to Phy, it is of the utmost importance to specify “ops.root” as the same folder of the DAT file to analyse, since Phy requires the original DAT file and the Kilosort results to be in the same folder.

Furthermore, some parameters concern the structure of the data and are mainly the same values contained in the channel map, which need to be specified also in the Kilosort configuration. Among these we have “ops.fs”, “ops.NchanTOT”, “ops.Nchan”, “ops.Nfilt” and “ops.nNeighPC”, which refer to the sampling frequency of the signal, the total number of channels present in the data (even those that will be left out of the spike sorting), active channels (channel that will be considered by Kilosort for the spike sorting), the number of clusters to use, and the number of channels to mask the PCs, respectively. The last two parameters depend on other prior ones and must be calculated according to the provided information. The maximum number of clusters to use to sort the spikes (“ops.Nfilt”) should be 2 to 4 times more than Nchan and a multiple of 32. However, for large datasets with few channels and extended time length, it can be desirable to increase this value over 4 times the Nchan, so as to provide more room for adequate sorting. The number of channels to mask the PCs is a criterion relevant in Phy visualization and, despite being noted in the source script that it can be left empty to skip, a value equal or lower than the number of active channels (usually around 3/4 of “ops.Nchan”) should be assigned, since leaving it empty leads to error in other fragments of the code.

At last, a few more parameters are related to the characteristics of the spike sorting method and thus affect the quality of Kilosort's outcome. The most relevant one, having distinct effects on the analysis, is the "ops.Th", which sets the threshold for spike detection in the analysis of the signal. For the data examined, belonging to human single neuron recordings of eight channels at a sampling frequency of 32kHz, the threshold with best quantity-quality trade-off in spike sorting was found to be [-8 -2 -2]. Changes in "ops.Th" alter the threshold on a per spike basis. The first number corresponds to the threshold spike detection used during the optimization and the second number (equal to the third) are used during the final pass. Both thresholds are applied to template projection, not to the voltage.

Once all the parameters in the "StandardConfig.m" file are specified, the last step is to introduce the correct file and folder paths in "master\_file.m" in order to begin with the analysis. At the beginning, both the folders containing Kilosort and npy-path but be added to the Matlab path. Then in "pathToYourConfigFile" the path to the working directory, where the "StandardConfig.m" is present, must be written. Finally, the code is executed running the "master\_file.m" file. The ordered steps present in the analysis are:

- Loading of raw data
- Computing of channel-whitening filters
- Application of filters to raw data
- Writing of whitened data and final preprocessing
- Optimization of templates
- Analysis of all found batches
- Final template matching pass

### **Manual refinind – Phy**

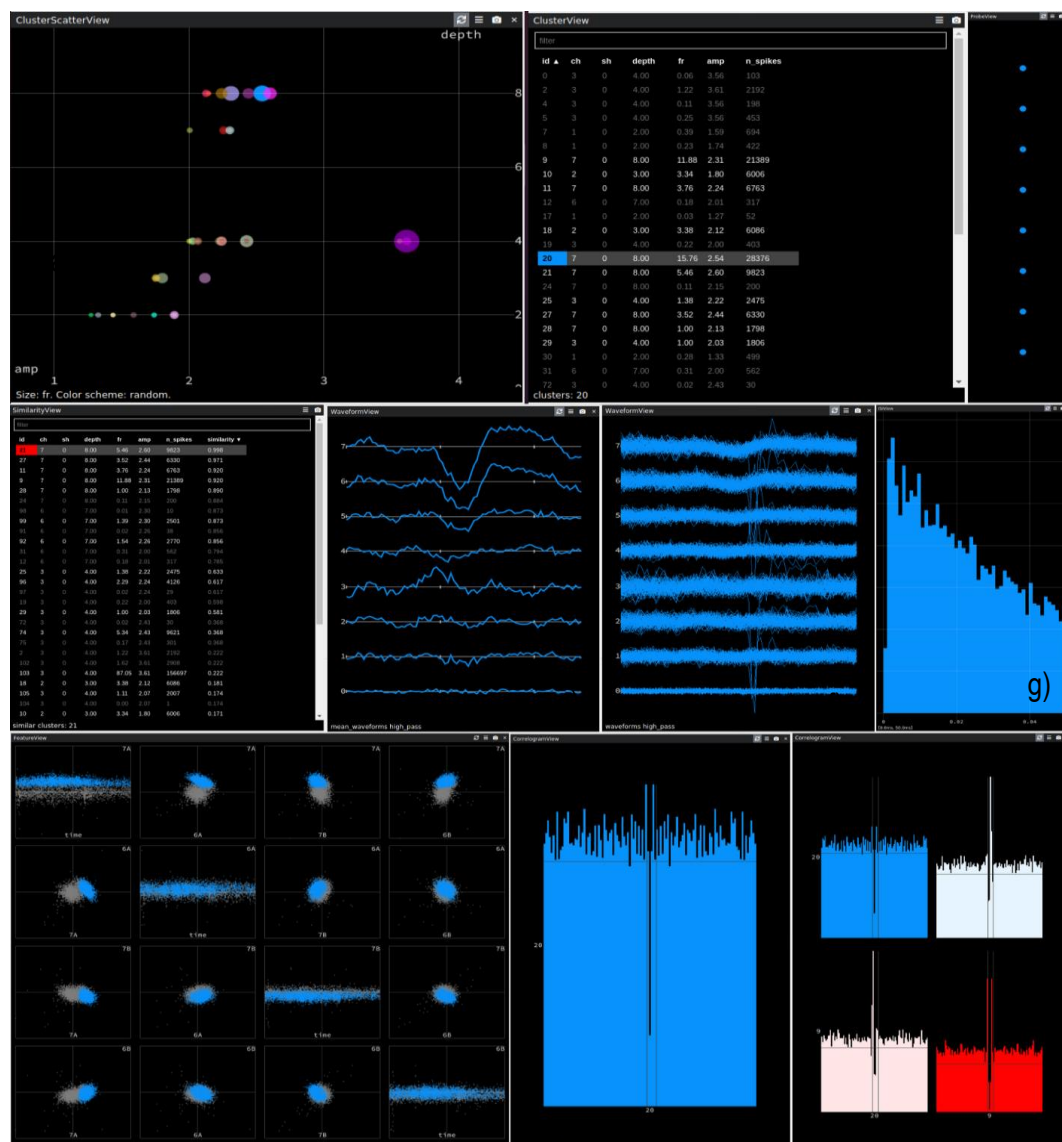
The objective of Phy is exclusively that of offering a series of tools for the user to manually curate the automatic spike sorting effectuated by Kilosort. In order to do this, a list of steps can be followed to ensure that all important information has been considered and the best decisions are taken. For this last method of the project, a complete pipeline with recommendations from the software's documentation, developers and other specialists is described, in order to obtain high quality results. This is also the complete set of instructions employed in order to analyse the data that is exposed in the Results section as an evaluation of the software, and also for additional research purposes undertaken by the developing team of this project.

Phy has two main requirements: a GPU to accelerate computationally intensive processes and a recent Python distribution (Phy Documentation). Therefore, it is implied that it would be suitable to carry out Phy manual refining in the same system that has hosted Kilosort spike sorting. It is specially so considering that Phy needs data from a variety of files created by Kilosort during the analysis, so avoiding data movement problems related to file loss can be prevented. The output of Kilosort is directly the data that is imported into Phy. It is of the utmost importance that the same folder (the working directory) contains all results from Kilosort and the original DAT file, otherwise Phy cannot render the data. Furthermore, Phy is executed from the system Terminal in Linux or

the Command Prompt in Windows. It is necessary to execute the commands in the working directory's path, in order to access the correct files.

The Phy repository must be downloaded from the Cortexlab GitHub page. Phy is free of use and open source, sharing many developers with Kilosort. Also, in case of needing the addition of a Plugin, it should be written and installed before opening the data. Before opening Phy, the command “`phy extract-waveforms params.py`” must be executed from the Terminal. This command generates the necessary waveforms in Phy, which otherwise would be vacant due to a discrepancy between Kilosort3 and Phy. Such discrepancy could be solved in future instances of the software or in other versions, consequently making this step unnecessary. Afterwards, the command “`phy template-gui params.py`” can be executed to start the Phy GUI.

The Phy GUI is composed of a series of windows and each one represents an aspect of the data.



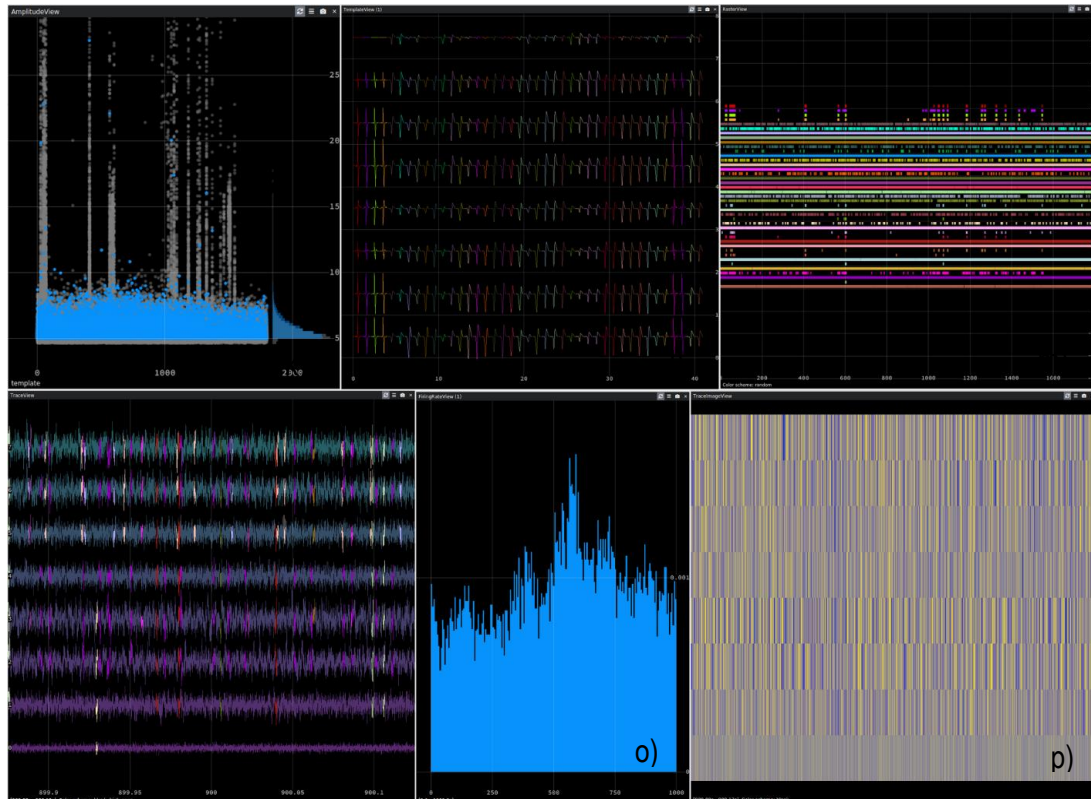


Figure 6. Every visualization available at Phy, representing data about a specific cluster or the entire dataset.

In Figure 6 all views available at Phy can be observed, each one involving a different feature of the data and with its own parameters and visualization (Phy Documentation):

- a) ClusterScatterView: represents all clusters in a scatter plot, as a function of the depth and the amplitude.
- b) ClusterView: shows the list of all clusters produced by the Kilosort sorting algorithm. Each cluster has a unique id number which only identifies that specific cluster; any modification in the spike contents of the cluster will lead to a new id being adopted. It also offers additional general information about the cluster such as number of spikes or channel and depth where the spikes of the cluster are more prominent.
- c) ProveView
- d) SimilarityView: visualization equivalent to the ClusterView, also showing a list of all clusters, but now they are distributed in order of similarity to the main cluster selected in ClusterView. It adds a similarity column, which indicates the degree of feature proximity between the two cluster.
- e) WaveformView: represents the waveform of the spikes present in the cluster. In this case, the average of all spikes in shows.
- f) WaveformView representing a large number of the cluster's spikes superimposed, instead of an average.
- g) ISIView or ClusterStatisticsView: histogram of the selected cluster, including inter-spike intervals and instantaneous firing-rate.
- h) FeatureView: visualization composed of different windows, presenting plots of the principal component features of the spikes. The selected cluster's spikes are marked in blue colour,

while the rest of spikes from every other cluster remains in grey. The diagonal windows represent each principal component in relation to time.

- i) CorrelogramView: shows the auto-correlogram of the cluster's spikes. The horizontal line marks the baseline firing rate and vertical lines identify a default refractory period of 2 ms.
- j) Comparison of two clusters in the CorrelogramView, with the white correlograms corresponding to the cross-correlogram between the two clusters.
- k) AmplitudeView: plots the spikes of a cluster (in blue) as a function of the amplitude and the timestamp, with all other cluster's spikes in grey.
- l) TemplateView: shows all templates of the session, and their position depends on the cluster order in ClusterView.
- m) RasterView: representation of all clusters in a raster plot. The order of the rows depends on the order of clusters in the ClusterView.
- n) TraceView: shows the raw signal marking in colours where and when each spike takes place.
- o) FiringRateView: histogram representing the firing rate of spikes in a cluster as a function of time.
- p) TracelImageView: represents the same information as in TraceView but as a textured image.

In the following part, a step-by-step guide is developed to manually refine Phy data using information from software manuals, advice from developers and specialists' instruction. The procedure is meant to offer a practical approach to Phy, where the trade-off between output data quality and time expenditure is balanced according to research standards. It represents a series of steps which are recommended to be taken in a specific order, so as to cover as many aspects of the software as possible. However, such description is only a navigation tool, which needs to be accompanied by a multifaceted approach to the data and preferably a brief training in order to increase the versatility and decision-making capabilities of the user.

In order to correctly handle the data, all windows need to be considered at different steps during the refining. For the study of these steps, the main documentation of Phy has been consulted, which already includes a manual for sorting in Phy and additional information about the GUI and the different parameters that can be modified. Furthermore, specialists in the field of single neuron recordings in mice were consulted for guidance, and supplementary tools and advice offered by the developers in several sources were also taken into account.

The following pipeline is meant as a series of ordered recommendations to be undertaken with the purpose of refining spike sorting output data from Kilosort with optimal results and time expenditure:

- 1) Identification and labelling of noise clusters.

Clusters containing predominantly or exclusively noise should be discarded right away for clearer classification of the remaining clusters. To identify a cluster as noise, several criteria from different views can be considered. Firstly, the general characteristics of the clusters can be taken into account to decide whether that cluster is considered, such as the number of spikes it contains. In case of small clusters, with under 1000 spikes, even if other aspects seem promising and indicate a possible neuron, it is usually unnecessary to keep that cluster because it cannot be used for



research purposes, due to the small magnitude. It can also be considered that low number of spikes leads to not useful statistical generalisation, so commonly in these cases most of the views are too irregular or look like stochastic processes, which makes them particularly hard to classify.

Then, its spatial and temporal distribution must be considered to identify sources of noise or artifacts. The presence of artifacts is signalled by high amplitudes and repeated firing concentrated at a precise time. Also, due to the refractory period a neuron cannot fire multiple spikes at the same time, so that accumulation of them at specific points in time can only represent noise. Hence, columns of high amplitudes in the AmplitudeView or vertical distributions of spikes in the diagonal windows (which represent each feature with respect to time) of the FeatureView indicate noise signals, as is represented in Figure 7.

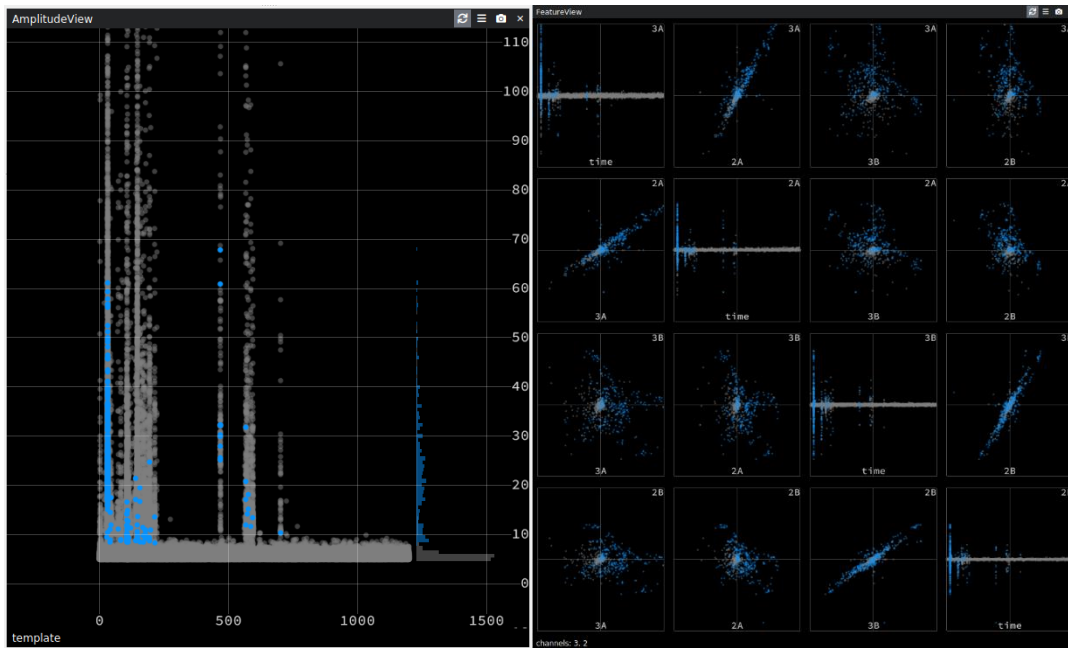


Figure 7. Example of an AmplitudeView (left) and a FeatureView (right) of a cluster containing noise, where vertical distributions represent large firings in the same time window, an indication of artifacts or similar disturbances.

Apart from artifacts and other visually significant alterations, some spikes can just be the result of random or periodical noise, in which case typical wave parameters must be analysed. The two most representative views are WaveformView and CorrelogramView. As mentioned, the refractory period is a basic element of any neuron which should be easily identifiable in an auto-correlogram, which should have a dip around 0, meaning that most of the spikes did not fire at the same time. However, it is also common to see a dip in the middle but with two high peaks at both sides of the 0, which indicate that the identified neuron could have a pattern of fast firing, releasing several spikes one after another in a short time span. In general, flat or anomaly-shaped correlograms are an indication that the selected cluster could be noise. An example of this effect can

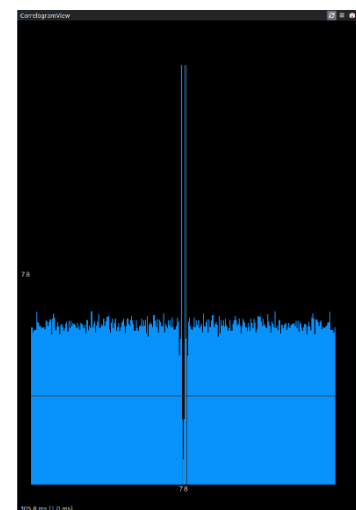


Figure 8. Example of mostly flat waveform, almost certainly attributable to noise.



be observed in Figure 8. Additionally, the cluster represented in this figure was composed of 104243 spikes, which supports the idea that it is composed mainly of noise.

On the other hand, the waveform of a neuron is very characteristic and should comply with a series of features to be considered a natural shape produced by an action potential. Generally speaking, the waveform is represented by a short depression followed by the refractory period (in many cases unappreciable). For spikes in different channels to be considered generated by the same neuron, their waveforms must have very similar shapes and usually different intensities. This constraint comes from the fact that the distance between microwires does not allow for many channels to record the same neuron, and thus any identifiable waveform shape present in many channels at the same time with the same intensity is most probably the result of artifacts or other higher-scale activity. An example of this situation can be observed in Figure 9. For clusters containing single neuron spikes, typically a characteristic waveform shape is present at highest intensity in one channel and at decreasing intensity in two or three more channels, with the rest showing alternative shapes or none.

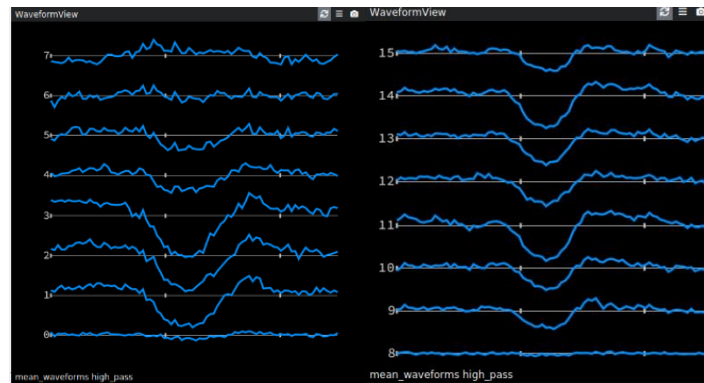


Figure 9. Phy WaveformView of a typical waveform shape (left) against a waveform with a regular shape but unnatural channel distribution (right).

## 2) Splitting clusters of different neurons and cleaning.

Splitting can be used to separate groups of spikes from a cluster, usually with the intention of dividing spikes belonging to two different neurons. However, splitting can also be used to remove noise-related spikes from otherwise acceptable clusters. Splitting is done manually drawing a polygon around the spikes that are meant to be separated (with the “Ctrl” key pressed and selecting each point of the polygon using the device’s mouse) and then pressing the “k” key. This drawing can be made in any view that represents individual spikes, like the AmplitudeView or the FeatureView, and thus in each window the parameter defining distance between spikes and spike distribution will vary.

In general, splitting a cluster into two different neurons is a complicated process that necessarily involves information from different views. The main tool employed in identifying clusters with possible multiple neurons is the FeatureView, in which data is distributed in several ways according to different features and also temporally. In case a cluster contained spikes from more than one neuron, in at least one of the windows in FeatureView this cluster would show two distinct groups of spikes, which could be manually separated. However, it is important to take into account that

FeatureView represents original data from spikes, while Kilosort implements filters to optimize the spike sorting process, so that it is reasonable to assume that the sorting effectuated by Kilosort is probably more complex and accurate, since it considers a wider array of variables and a higher number of dimensions. For this reason, splitting a cluster into two neurons is often discouraged.

However, in case a splitting is intended for two groups of spikes in the same cluster, additional views can be considered to make this decision. Basically, if two groups of spikes are already separated by a feature in FeatureView, then characteristics of the possible neuron they each relate too should be considered. Should they belong to different neurons, the WaveformView of both of them would most probably show different distributions and waveform shapes. Similarly, in the CorrelogramView, the cross-correlogram of both groups can be studied to analyse their differences. A cross-correlogram with a central dip, similar to the auto-correlograms of each, would indicate that they are actually the same neuron. Otherwise, a flat cross-correlogram means that there is no relation between the firing patterns of both groups of spikes, and thus belong to separated neurons.

On the contrary, splitting can also be used to clean a cluster and remove spikes clearly related to noise, while keeping otherwise normal spikes (possibly related to neurons), such as in Figure 10. In this case, the two most useful visualizations are again AmplitudeView and FeatureView, where data is represented with respect to time. Vertical cumulations of spikes in the AmplitudeView, characterized by large amplitudes, and similar distributions in the time windows of the FeatureView, in which many spikes are present at the same time, are clear noise situations, as described in the above section. Therefore, the splitting tool can be employed to separate these cumulations without removing the low amplitude spikes present in the same time range, which can be attributed to normal action potentials. This technique can help during the classification of complicated clusters, but at the same time is burdensome and time expensive. For this reason, this step is generally unnecessary in cases of abundant data, where such inconclusive clusters are discarded or counted as MUA, considering that later in the analysis only the neuron spikes will be relevant.

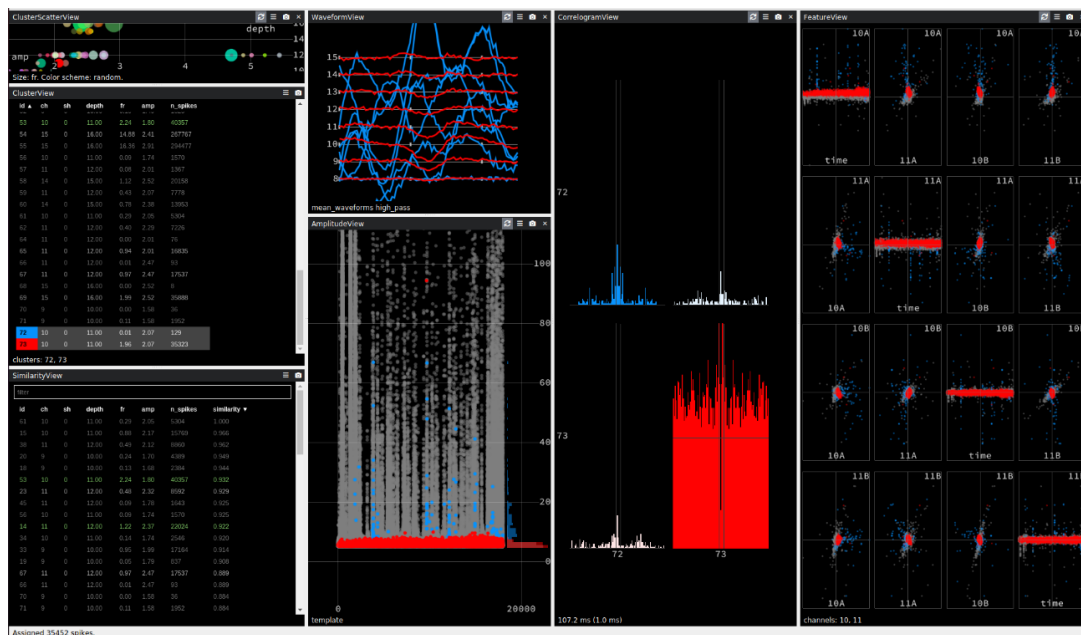
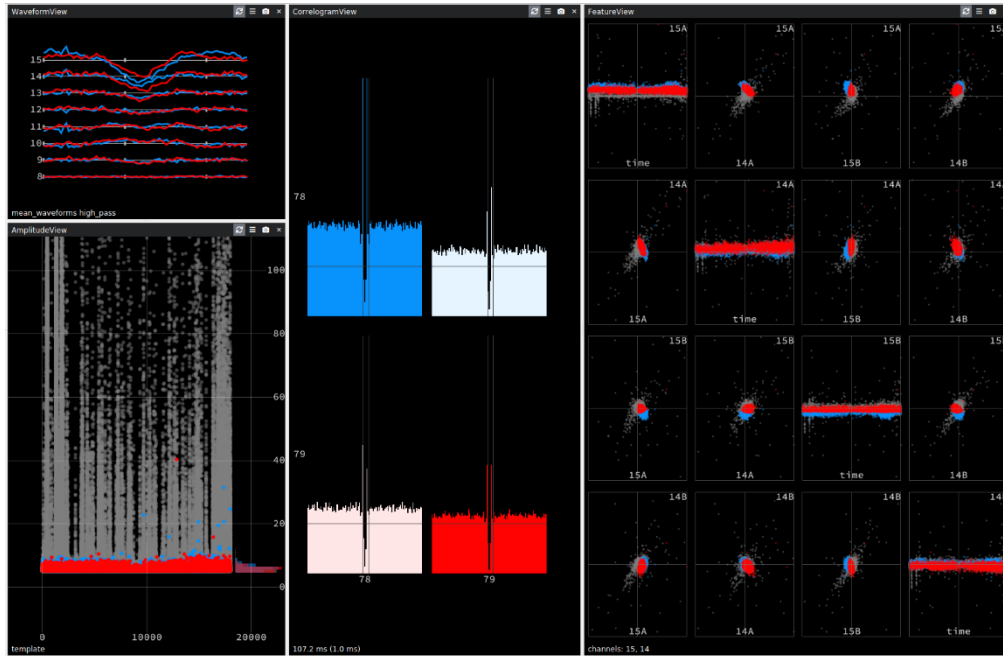


Figure 10. Example of a splitting procedure applied to a cluster to remove vertical columns in AmplitudeView, dividing it into two new clusters. The most significant windows represented are the ClusterView, AmplitudeView, CorrelogramView and FeatureView.

### 3) Merging clusters

In some cases, Kilosort can separate into different clusters spikes belonging to the same neuron. Therefore, it can be useful to merge such clusters together. Merging is done by selecting the two desired clusters in the ClusterView and pressing the “g” key (for grouping).



*Figure 11. Parallel representation of two clusters with equivalent characteristics, suggesting that both belong to the same neuron and should be merged.*

Before merging, however, it is important to compare similar clusters. In order to do so, Phy presents the SimilarityView, which has the main purpose of encompassing the comparison of all analytic parameters between two clusters in a “similarity” value, which goes from 1 (all parameters match) or 0 (there is no similarity between the clusters). Using this tool, for each cluster the most similar options can be considered, comparing the aforementioned behaviour in WaveformView, CorrelogramView and FeatureView to decide whether their spikes belong to the same neuron. As can be observed in Figure 11, cross-correlograms with clean refractory periods indicate no spike overlapping and thus a high probability of having the same neuron, as well as similarly shaped waveforms. Similarly, overlapping of two clusters in most of the FeatureView windows indicates that they have many features in common in several dimensions. Overall, many parameters have to be considered collectively to determine whether two clusters are suitable for merging, as is the case for Figure 11. In cases of having resembling waveforms but one of them is clearly more chaotic, one could be dealing with multiunit clusters, in which case it would be better to not merge and be classified as MUA. Usually, similarity values below 0.6 are regarded as to not having enough correlation, so they can be excluded of the comparison or can be included at the expense of longer curating times.

Some other cases that could require merging include Drifting and Burst adaptation, both graphically exemplified in Figure 12. Drifting consists in the displacement of the electrodes during the recording period, so that spikes produced by the same neuron are detected by different channels at two

different time ranges (or more if several electrode shiftings occur during the recording). This phenomenon, which Kilosort would automatically sort into two separate clusters, can be easily observed in the AmplitudeView or the FeatureView, where each cluster would only have spikes present in a portion of the recording, but represented at the same time they would cover the whole period. Also, their cross-correlogram should perfectly match the individual auto-correlograms.

Burst adaptation, on the other hand, is created by adaptative firing in neurons, which produce different types of spikes due to the dynamic characteristics of their waveforms. As such, Kilosort also sorts them into different clusters. They are more difficult to detect, and several parameters should be considered. Firstly, the AmplitudeView, when the two clusters are compared, would show spikes from both of them during the entire length of the recording, but at different amplitudes, so that they are separated in the y axis of the representation. Also, the waveforms would have a similar shape but with different amplitude and extensions, and the WaveformView would show both of their waveforms in the same channels. Finally, the cross-correlogram between the two would be highly asymmetric, with one side being mostly flat at 0. If such case arises where all these parameters match the description, it can be concluded that despite the different waveform characteristics they belong to the same neuron and a merging can then be undertaken.

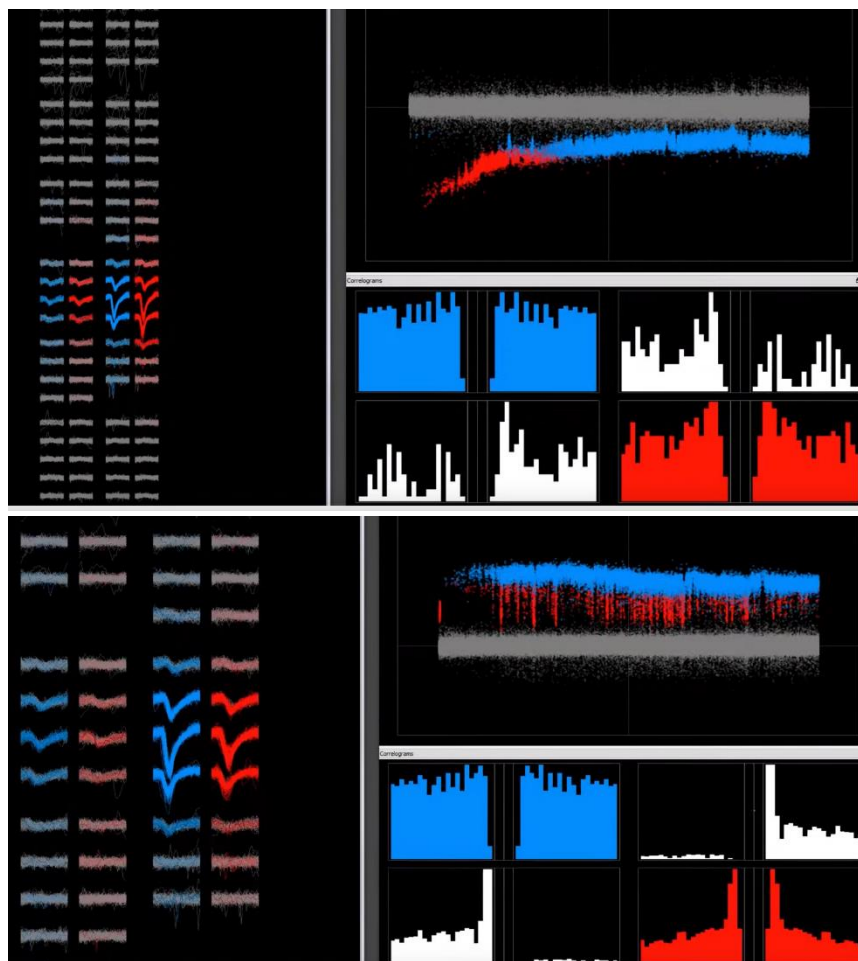
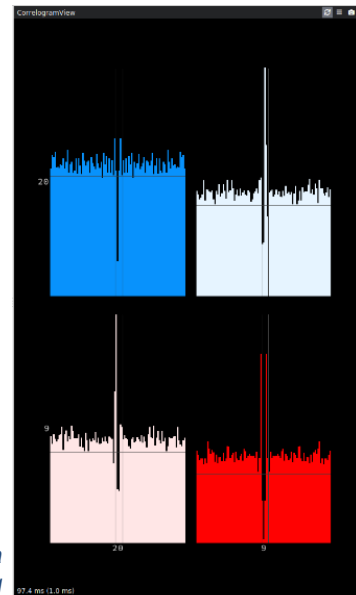


Figure 12. Images of drifting (top) and burst adaptation (bottom) from 2020 Lecture 2.05 - Using Phy to curate spike sorting by Nick Steinmetz (UW) (YouTube). The views represented are WaveformView, AmplitudeView and CorrelogramView.

Regarding the asymmetrical cross-correlograms in clusters generated by burst adaptation, a similar effect can be observed in clusters containing independent neurons where the cross-correlograms are mainly flat on all sides except on the centre, which is divided in a depression similar to a refractory period and an elevation. This asymmetry, however, is limited to the central part of the correlogram, so it has no connection with burst adaptation, but it is often interpreted as a sign of two independent neurons linked, that is firing one after the other. For this reason, the cross-correlogram identifies a firing correlation in a short time difference, while other wave parameters are unrelated.



*Figure 13. Example of a cross-correlation which presents an asymmetry, with a possible interpretation being that the two clusters are two connected neurons firing*

Nevertheless, it must be noticed that, similarly to what was described in the splitting process, merging is in every case a nonessential procedure, which can be decided not to be applied. This decision can be made contemplating a variety of reasons, the most important ones being the uncertainty of merging judgement, the lack of added value for posterior analysis and the fact that it is severely time-consuming. Alternatively, this procedure can be omitted for lack of clear classification metrics, especially in cases where future analysis of the data will employ more advanced statistical methods (such as in spike correlation between neurons). Then, the later analysis can help clarify the uncertain sorting of specific clusters, which can lead to a second cycle of Phy labelling before the results are purposed for scientific or clinical research.

#### 4) Difficult decisions and final labeling

The last step for data curation in Phy resolves around using all prior steps described to tackle difficult decision and enact the final labeling that will be used for subsequent studies. In many cases, this labeling can be inconclusive, in order to not risk losing information, at the expense of having less defined results. Such results could be enough for high-input research or statistical studies, in which the presence of marginal noise is systematically rendered inconsequential, and sorting anomalies such as unnatural division of single neuron spikes or multiunit provide useful and label-appropriate information.

## Troubleshooting and alternative steps

### Troubleshooting and error solving

During the previously described methods, several situations are prone to missteps which lead to errors. Such errors usually arise from the software system employed, either Matlab or Python, and can encompass many libraries and nested scripts, making them difficult to identify.

In Matlab, due to the development of the entire script and the use of especially precise libraries, if followed correctly there is little window for errors. It is important to ensure the data introduced is in

the appropriate format (NCS Neuralynx file) and that any parameters regarding structure of the data are modified only in its definition at the beginning of the script, to avoid inconsistencies. The whole script is designed to be mostly automatic and self-regulatory, so that values inside the loops should not be changed. However, in case of processing alternative data, such as different file types or data structures, the main parameters can be modified without risk of encountering errors. This is mainly the case for channel structuring using Fieldtrip, since it is capable of reading a wide array of file formats and importing the data to a standard configuration, and notch filtering, which has been tested with a variety of statistical parameters to adapt to each situation and peak detection as well as attenuation can both withhold a broad range of values.

On the other hand, Kilosort's algorithm is more complex and involves more levels of computationally intensive functions, so small digressions from the described steps can readily turn into errors. The following lines of text are an example of the kind of an error produced by Matlab while running Kilosort:

*Error using /  
Not enough input arguments.*

*Error in **preprocessData** (line 69)  
ops.sampsToRead = floor(d.bytes/NchanTOT/2);*

*Error in **master\_file** (line 19)  
[rez, DATA, uproj] = preprocessData(ops); % preprocess data and extract spikes for  
initialization*

The first line describes the error, with a more or less identifiable explanation, and the rest indicates which files and functions are affected, in order to pinpoint the source of the malfunctioning. In this case, the error originates as a result of an incorrect introduction of the DAT file's path in the "StandardConfig.m" file (in the variable "ops.fbinary"). A similar error is produced when, in the same file, the variable "ops.nNeighPC" is left blank or as a "null". For this reason, it is important to assign a numerical value below the total number of active channels.

Another example of a Kilosort error as observed in the Command Window of Matlab:

*Error using **gpuArray**  
Encountered unexpected error during CUDA execution. The CUDA error was:  
CUDA\_ERROR\_ILLEGAL\_ADDRESS*

*Error in **fitTemplates** (line 198)  
dataRAW = gpuArray(dat);*

*Error in **master\_file** (line 20)  
rez = fitTemplates(rez, DATA, uproj); % fit templates iteratively*

This error can appear in several circumstances and is related to CUDA. Experimentally, this error has been found to appear in successive executions of Kilosort, without seemingly altering relevant parameters, and probably has no connection to the files used as input for the software. The most

basic course of action in this case is to restart Matlab, which solves the error without requiring any additional manipulation, and Kilosort can be used again regularly.

The standard GUI of Phy has a limited array of options, which limits the type of actions that the user can perform. Hence, it is hardly possible to encounter errors while working with the basic Phy interface. The main two situations that can lead to malfunction of the software are no prior extraction of waveforms and defective plugins. As mentioned before, executing the command “phy extract-waveforms params.py” in the system’s console is necessary before working with Phy in order to have access to its full functionality. Otherwise, some option will not be present and trying to generate specific visualization windows will lead to error. On the other hand, since plugins are independent from the main software and created by users, they can interact in many different ways with Phy and could cause internal inconsistencies which would have a detrimental effect on basic functions, thus creating errors in the GUI.

### **Alternative steps**

The decision to employ Kilosort 1 instead of more developed versions was supported by recommendations from developers and a deep understanding of our data. However, as recording devices improve and human neural activity studies evolve, new technologies could soon allow the obtention of higher quality signals, closer to the type of recordings that can be collected nowadays in mouse research. Hence, in a near future later versions of Kilosort could prove more useful and advanced in human single neuron spike sorting. For this reason, in this section it will be explained how to perform the same analysis described before but in Kilosort3, the latest update of the software.

Kilosort3 was first developed in 2018 and includes several improvements with respect to the prior versions. Compared to Kilosort 1, the main difference is the presence of a GUI, which allows the modification of analytic parameters directly on the interface and the visualisation of channel data prior to the spike sorting. All functions described in Kilosort 1 are now accessible either by changing specific parameters in the GUI or by handling the graphical representation of channels (such as channel inclusion or exclusion, which can be selected directly on the visualization window). Compared to other versions, Kilosort3 improves the drift correction introduced in Kilosort2 and introduces a new more sophisticated clustering algorithm.

Similar to Kilosort 1, Kilosort3 must be executed by running the “kilosort.m” file present in the main folder to the repository. Matlab automatically opens the GUI, and in case it has been used previously it resumes the files and configurations set. Otherwise, using the “Select data file”, “Select working directory” and “select results output directory” buttons, the necessary paths and files must be selected. Also, in the “Select probe layout” drop-down, the channel map can be browsed from a selection of predefined distribution, or a previously created channel map file can be added using the option “other...”. Additionally, it is possible to create a channel map directly in the GUI, by selecting the option “[new]”. In this case a new window opens for the user to introduce the channels’ parameters, which are the same as described for Kilosort 1.

The rest of necessary parameters are to be written in the respective blank box below. It is important to introduce correctly the “Number of channels” and the “Sampling frequency (Hz)” as those are



necessary to correctly read the data. Moreover, “Threshold” can also be selected, which is by default set to “10 4”, as well as other determinants of the analysis which are however generally unchanged, like “Lambda” and “AUC for splits”. Finally, as mentioned above, the active channels which will be considered later for the spike sorting can be selected directly in the “Probe view” or the “Data view” by right clicking the respective representation. Also, the fragment of the data to analyse can be truncated by changing the upper and lower limits set in “Time range (s)”, which defaults to “0 inf”.

The procedure is the same and contains roughly the same steps as in Kilosort 1, resulting in parallel output files which have equal characteristics adapted to Phy visualization. However, the GUI makes it more difficult to identify new errors which are easier to appear, due to more complex processes taking place during the spike sorting. The main ones and how to solve them are explained below.

Kilosort stopping at the “Computing whitening filter” step indicates problem at the data importing level and are related to the format of the data or the channel map. The situation where this issue is most prevalent is in cases where the data has not been adequately formatted. For this reason, it is very important to use correct functions in the preprocessing of the signal. Data needs to be encoded as binary integer 16 in a DAT file, which for the case of Matlab is optimally done using the “fwrite” function.

Afterwards, if the analysis stops at the “Preprocessing” step, it can have a wider array of possible explanations. A common error is related to the shift correction algorithm introduced in this version of Kilosort. Different data interacts in a variety of ways with this algorithm, and some can lead to errors. However, the files affected are usually printed in the Matlab Command Window, where an analogous error to the one appearing in Kilosort is described. An example would be a description similar to:

*Error: a interp1, "Sample points must be unique."*

Then, the best course of action is to deactivate the drift correction for this session, changing the “ops.nblocs” parameter in “main\_kilosort3.m” from 0 to 5.

Finally, a particular error can occur in the the “Extracting spikes for clustering” level, which gives out the following message in Matlab:

*Error: a gpuArray/subsasqn, "An unexpected error occurred trying to launch a kernel. The CUDA error was: invalid configuration argument"*

In this case, there are two files that need to be modified. First, in the “extract\_spikes.m” file (inside the “clustering” folder of the repository) a try/catch function must be added surrounding line 97. If only this correction is made, still problems arise in the “First clustering” step, so also the “template\_learning.m” file (in the same “clustering folder) must be added another try/catch function in line 71. This way, Matlab avoids certain paths that would lead to inconsistencies in the sorting.



### 3. Results

The main results of the project are obtained at the end of the entire pipeline, when the files containing information about spike firing and sorting into neurons is produced by Phy, which is the ultimate goal of the software. However, it is composed of several methods divided into individual steps, each of which is meant to modify the data in a specific way or add a layer of pre-processing. Considering every step as its own process, the results of each method can be quantitatively and qualitatively analysed and evaluated to ensure optimal functionality and efficiency.

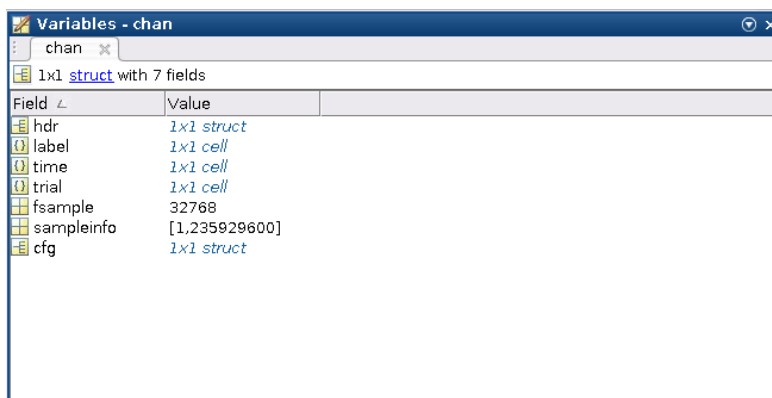
In the following sections, the individual results of each method will be exposed, along with comparisons of similar results obtained applying differential parameters or calculations.

#### Pre-processing and filtering

Since the steps involved in pre-processing and filtering have been developed for this software and must prepare the data for the ensuing analysis, it comprises a series of complex tools intended to optimise the quality of the signal. To do so, Matlab loops are implemented in order to generalize the procedure and make it more autonomous, while applying external Toolboxes for the tackling of specific tasks.

As mentioned before, data must be imported from CSC files (Neuralynx). The Fieldtrip toolbox makes this operation possible, by introducing the path of the file. Then, all the data in the file is imported in as a Matlab *struct* (structure), a type of variable which can include subfolders to organize data. As can be observed in Figure 14, the information of the signal is stored as five sub-variables:

- “hdr”: header of the file
- “label”: name of the signal’s channel
- “time”: list of numbers with the timestamps of the entire recording
- “trial”: containing the signal data
- “fsample”: frequency sample
- “sampleinfo”: information regarding dimensions of the data
- “cfg”: Fieldtrip parameters used in the importing process



Field	Value
hdr	1x1 struct
label	1x1 cell
time	1x1 cell
trial	1x1 cell
fsample	32768
sampleinfo	[1,235929600]
cfg	1x1 struct

Figure 14. Example of structure containing data from an CSC file.

Then, the most important part of the Matlab script is the filtering. In order to remove noise frequencies, a notch filter from DSP Toolbox is implemented. The notch filter is applied to every signal recording using independently calculated parameters. Such parameters depend on statistical calculation of the signal as well as the data fragment surrounding the notch frequency. A function using the difference between notch frequency power and median power of a  $\pm 3000$  samples window, the average power in the same window, and the frequency of the notch frequency was decided on as the best-performing combination of parameters, with a constant parameter which should be altered for entire datasets to tune attenuation intensity.

The main challenges faced were the lack of attenuation and the disproportionate attenuation of notch frequencies in the same signal according to frequency or spectral power. In order to supply a balanced algorithm that would take all of these aspects into account, several statistical factors had to be considered. Firstly, an uneven distribution of attenuations between initial and posterior frequencies was observed. For this reason, the frequency of the notch frequency was included in the formula. Then, most importantly, the different between the median power and the power of the notch frequency is directly proportional to the attenuation that must be applied to that frequency.

Additionally, it must be noted that noise frequencies were not only determined by one single spike at a specific frequency but were surrounded by series of high-power frequencies. For this reason, part of the looping was dedicated to identifying only the highest spike in a cluster of high-power frequencies, to determine the notch frequency that was meant to be filtered. Furthermore, this accumulation of high-power frequencies led to elevated values of average power around the notch frequency, even with large sample windows, for this reason the formula includes both the average power and the median power in different places, to obtain the best context-dependant performance according to the specific characteristics of each value. The median was generally lower and closer to the baseline power.

In Figure 15, the first 273.0665 Hz of the same signal are represented before and after the implementation of the recurring notch filter, named "Pre 2-1" and "Post 2-1" respectively. The first plot of each representation (top) belongs to a simple Matlab 2-D line plot representing spectral power as a function of the frequency. The second plot (bottom) represents the same data using a semi-log plot, in order to better discern differences in power. It can be observed how several noise frequencies showed high-power spikes in the pre-filtered data. After the implementation of the filter, all noise frequencies are removed with an attenuation equivalent to the original spike's spectral power, so that no visible spikes remain while not altering significantly the surrounding frequencies.

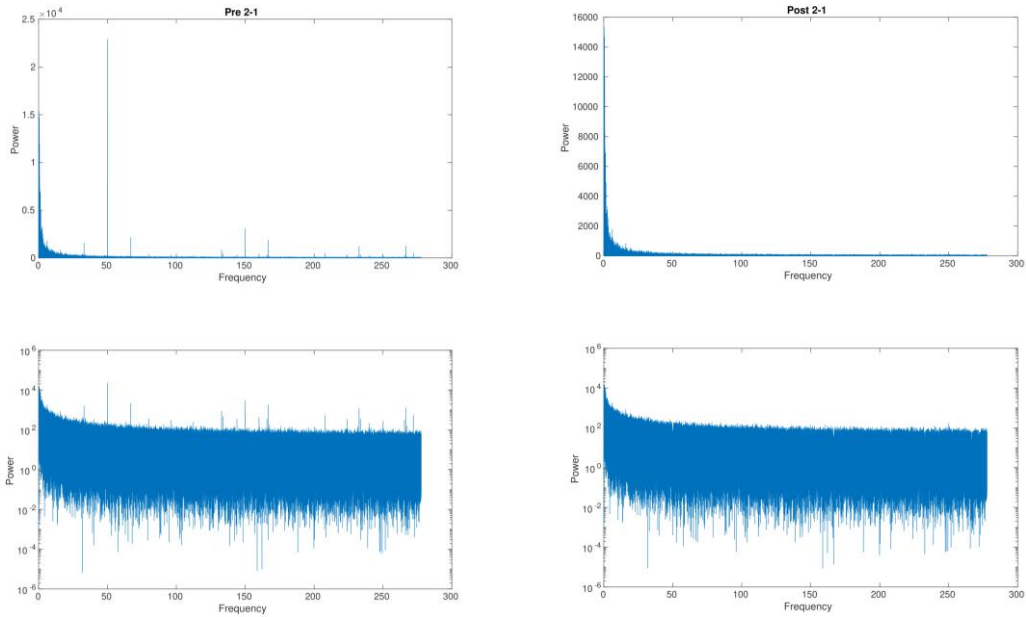


Figure 15.. Example of fragment before (left) and after (right) the notch filter, using Matlab functions plot and semilogy.

Afterwards, the last steps in the pre-processing are directed towards unifying all filtered data into a single *struct* variable, with the distribution and characteristics of the original *struct* created by Fieldtrip, in order to exported in the desired binary “int16” DAT file format which is required by Kilosort. The main differences between the final *struct* and the original one from Figure 14 are the “label”, “trial” and possibly “time” categories. The “trial” array will have added as many rows as channels have been concatenated. All channels must have the same sample frequency and length. Additionally, trial might have also added more columns, in case consecutive fragments of the data have been added together. The same situation of fragment concatenation is the only case in which “time” can change, with the added columns after the initial timestamps list. It must be noted that despite including several channels in “trial”, all of them have the same timestamps, since the sample frequency and the length do not change, so “time” will remain being composed of only one row. Finally, “label” will have increased from a single cell containing the name of the source channel to an array of label names where the rows represent channels, and the columns represent fragments. Moreover, different fragments of the same channel will not have the same label name if they come from different recordings, as is the case for Figure 16, where several channels from separate whilst consecutive recordings of the same individual have been merged a *struct* which be exported as a single output DAT file.

data_whole.label					
	1	2	3	4	5
1	02718_2019-05-21_23-01_mTbm_1	02718_2019-05-22_01-01_mTbm_1	02718_2019-05-22_03-01_mTbm_1	02718_2019-05-22_05-01_mTbm_1	02718_2019-05-22_07-01_mTbm_1
2	02718_2019-05-21_23-01_mTbm_2	02718_2019-05-22_01-01_mTbm_2	02718_2019-05-22_03-01_mTbm_2	02718_2019-05-22_05-01_mTbm_2	02718_2019-05-22_07-01_mTbm_2
3	02718_2019-05-21_23-01_mTbm_3	02718_2019-05-22_01-01_mTbm_3	02718_2019-05-22_03-01_mTbm_3	02718_2019-05-22_05-01_mTbm_3	02718_2019-05-22_07-01_mTbm_3
4	02718_2019-05-21_23-01_mTbm_4	02718_2019-05-22_01-01_mTbm_4	02718_2019-05-22_03-01_mTbm_4	02718_2019-05-22_05-01_mTbm_4	02718_2019-05-22_07-01_mTbm_4
5	02718_2019-05-21_23-01_mTbm_5	02718_2019-05-22_01-01_mTbm_5	02718_2019-05-22_03-01_mTbm_5	02718_2019-05-22_05-01_mTbm_5	02718_2019-05-22_07-01_mTbm_5
6	02718_2019-05-21_23-01_mTbm_6	02718_2019-05-22_01-01_mTbm_6	02718_2019-05-22_03-01_mTbm_6	02718_2019-05-22_05-01_mTbm_6	02718_2019-05-22_07-01_mTbm_6
7	02718_2019-05-21_23-01_mTbm_7	02718_2019-05-22_01-01_mTbm_7	02718_2019-05-22_03-01_mTbm_7	02718_2019-05-22_05-01_mTbm_7	02718_2019-05-22_07-01_mTbm_7
8	02718_2019-05-21_23-01_mTbm_8	02718_2019-05-22_01-01_mTbm_8	02718_2019-05-22_03-01_mTbm_8	02718_2019-05-22_05-01_mTbm_8	02718_2019-05-22_07-01_mTbm_8
9					
10					
11					
12					

Figure 16. Example of “label” sub-variable in output struct, containing a series of channels with several fragments concatenate, each of them imported from a different source file with an identifying label name.

## Spike sorting

Kilosort compiles a series of steps to process the data and extract the appropriate information. However, since all of the procedure is undertaken in a single execution, none of the intermediate results can be visualised, only the resultant output. On the other hand, variations in analysis parameters greatly affect the results of Kilosort, and as such a series of trial-and-error processes were implemented during the study of many of them in order to determine the optimal value.

One of the most important inputs of Kilosort, which needs to be precisely defined and attuned to the data characteristics, is the channel map. Most of its content are list of values which can be automatically defined by the “createChannelMapFile.m” file included in the software repository. Manual generation or modification should take into account the presentation of such lists and the type of variables that they contain. A particular example is the “kcoords” element, which appears to contain 1 and 0 values, but they are simply representations of *true* and *false*, the adequate variables for this parameter. However, for the case of human single neuron, the most difficult element to consider is the geometrical distribution of channels, defined by “xcoords” and “ycoords”. Despite the employment of Kilosort 1, which is supposed to minimise the impact of channel organization on the analysis, a comparison of different models was still undertaken in order to ensure sorting stability.

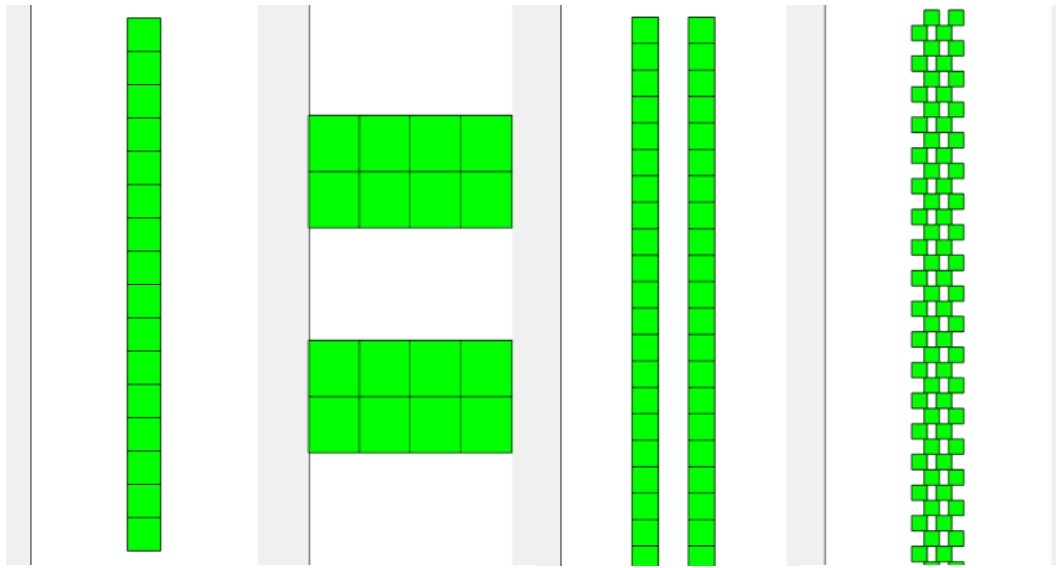


Figure 17. Different micro-wire distribution as observed in the Kilosort GUI.

Among the many options available, since according to the data analysed there is no actual limitation regarding position and even distance is simply uncertain, the most common geometries were attempted. In Figure 17 some channel map distributions are shown. The first two are geometries specifically tried during our project, finally deciding of the leftmost one, while the last two are example geometries included in the source Kilosort repository.

Additional parameters which have an impact in the analysis are the “ops.Nfilt” and the “ops.Th”. Nfilt refers to the limitation of clusters into which spikes are sorted. It is recommended to be set to 2 to 4 times the number of channels to process (Nchan). However, in cases of long recordings with few channels, where large amounts of spikes are extracted, it would be useful to increase the value

of Nfilt in order to give more room for spike classification. In an example carried out with a large whole-night dataset containing only 8 channels, with Nfilt=32 most of the 32 clusters produced included vast numbers of spikes and were exceedingly general, so that barely any neuron cluster was identifiable. Increasing Nfilt to 64 or even 128, led to more diverse clusters, with a range from a dozen spikes to several hundred thousand, so that noise, multiunit and good neurons were more easily discernible.

On the other hand, “ops.Th” contains the variables used in spike identification and extraction, working as an amplitude threshold. The default Kilosort threshold is set to [10 4] or [10 4 4], with the second and third number referring to the same value. However, this threshold has been found to be excessively high for the type of recordings that were analysed, so it was decided to lower it. The idea was to keep an appropriate distance between the two values, while alternatively changing both of them to obtain diverse results. The output of each threshold analysis was then observed in Phy. The table in Figure 18 represents all thresholds attempted, as well as some features of the results as studied in Phy, superficially scanning the dataset to decide on a few probable neurons (without following the whole refining pipeline described before, in order to limit the time expenditure, considering only clusters with good waveform, spatial and temporal distribution, and refractory period).

Threshold	Total number of clusters	Number of spikes in largest cluster	Number of probable neurons	Average number of spikes in selected probable neuron clusters
[0 -6 -6]	31	6488	2	2990
[0 -8 -8]	31	2946	0	0
[2 -4 -4]	31	35323	5	5361
[-4 2 2]	31	611541	1	123957
[-4 -12 -12]	31	1961	0	0
[-6 0 0]	31	220671	5	8120
[-8 0 0]	31	33425	4	7355
[-12 -4 -4]	29	14685	3	1777
[10 4 4]	31	60330	2	8460
[-8 -2 -2]	31	23365	5	6620
[-6 -2 -2]	31	23485	5	5850
[-8 -4 -4]	31	119319	2	5097

Figure 18. Table with various numerical features about the Kilosort results obtained using each of the evaluated threshold values.

The thresholds with best results are marked in coloured bands. The main parameters to establish a threshold as good is good classification power, identifiable in the number of probable neurons that were identified. The number of spikes present in the largest cluster is also interesting, as it can be considered an estimate for the general number of spikes obtained with that threshold. Large numbers of spikes are usually desirable, but without incurring in extremely high values, in which case it would be mostly composed of noise and no significant results would be obtained (such as

in the  $[-4 \ 2 \ 2]$  threshold). It can be considered that having more spikes makes classification easier, since most of the parameters in Phy are statistical and thus larger populations obtain clearer visualizations.

Similarly, the average number of spikes in the selected probable neuron clusters offers best results when it has significant dimensions, in the order of several thousand. At the same time, however, it is interesting to have neurons not only in big clusters but also smaller ones. This is because larger clusters have a higher tendency of accumulating noise, so that the more spikes a cluster contains, the most probable a large percentage of it could be noise. So having only large clusters makes for more insecure classification.

In the end, thresholds of first values around  $-6$  and  $-8$  showed good results, paired with second values closer to positive numbers, like  $-2$  or  $0$ . Qualitatively,  $[-8 \ -2 \ -2]$  was chosen due to better clarity of clusters (more defined waveforms shapes and refractory periods, for the case of the individual recording considered). Nevertheless, this analysis is remarkably dependant on the source data (optimal values would widely change from one sample to another), so this was chosen to simplify procedures while still obtaining good results.

Ultimately, the files produced by Kilosort are stored in the selected folder alongside the source DAT file containing the raw data, which will also be necessary for the Phy analysis. Each file contains different information regarding the sorting effectuated by Kilosort, such as the number of spikes, their characteristics, timestamps, label, etc. The necessary files for manual refining are automatically detected and opened by Phy, so there is no need for accessing any of them after the automatic sorting. However, in some situation it might be useful to use the results structures obtained from Kilosort, saved in the “rez.mat” file, a Matlab type of file which encodes a series of variables stored together. It must be opened with Matlab and includes several important parameters which can be analysed (Kilosort Documentation):

- Information regarding channel distribution and contents, mostly the same parameters defined in the channel map, such as “xcoords”, “ycoords” or “connected”.
- “st”: composed of four columns, with each column expressing one characteristic of each spike, namely the length of the spike in samples, the template matching spike shape, the extracted amplitude, and the final cluster in case of applying auto-merging.
- “mu”: the mean amplitude of each template used.
- “ops”: with all the configuration used in the automatic clustering.
- “Wrot”, “WrotInv” and “Wraw”: containing information about Kilosort pre-processing and filtering in the whitening matrix.
- “U” and “W”: low-rank components of the temporal and spatial masks applied to each template. They are the low-rank decomposition of “dWU”, consisting of a subset of spikes for each template.
- “simScore”: correlation between all pairs of employed templates.
- “cProj”, “iNeigh”, “cProjPC”, “iNeighPC” and “cProjPC”: regarding information about principal components of each template and spike projections.

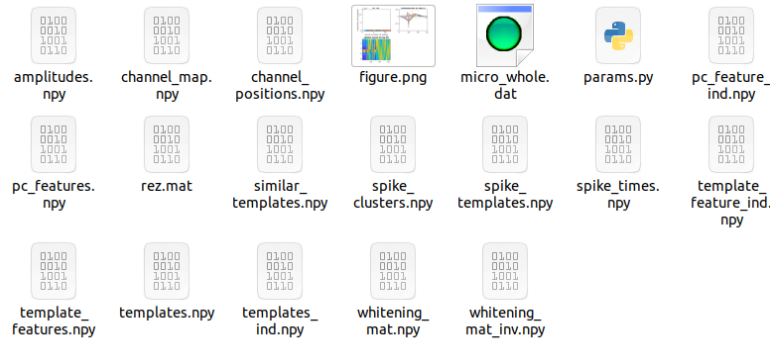


Figure 19. All files produced by Kilosort 1, as observed alongside the source DAT file.

## Manual refining

As opposed to Kilosort, the procedure in Phy is completely manual and thus every step of the process can be observed and analysed. As mentioned above, the main actions that can be undertaken in Phy are splitting and merging. However, splitting of clusters is uncommon, since evident differentiations between spikes are mostly already detected by Kilosort, so even specialized observation rarely leads to splitting. On the other hand, merging is a time-consuming process which is often left out of the method, unless for exceptionally obvious same-neuron clusters, in favour of other cleaning techniques and post-processing, which can mostly determine and correct unmerged clusters. The main step that is regularly pursued to obtain better visual results and clean the data is the removal of parallel high-amplitude spikes, which are commonly related to noise and artifacts. In Figure 20, a typical procedure can be observed, where such distanced spikes are isolated first from the FeatureView and then from the AmplitudeView. The results show a quality improvement in many of the available views, such as in the WaveformView, where waveform shape is more natural and following a more regular pattern. More remarkable is the change in the auto-correlogram, which presents a more interesting shape, with the indication of a possible refractory period at the center and thus is closer to be considered a candidate for a single neuron cluster, whereas the initial auto-correlogram would probably be discarded as noise right away.

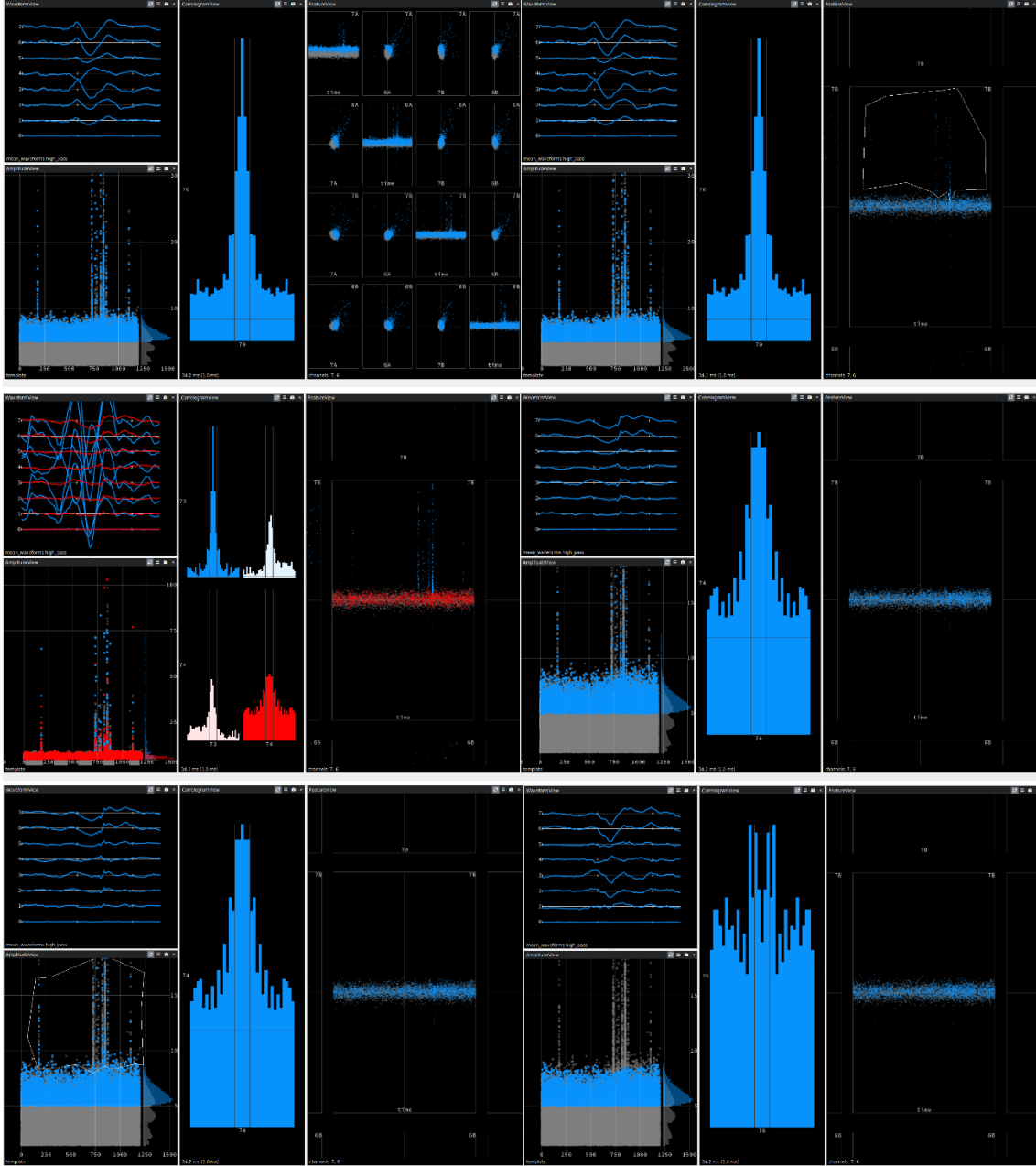


Figure 20. Images of a regular procedure to eliminate artifact noise using mostly FeatureView and AmplitudeView.

Apart from manually cleaning and improving signal-to-noise ratio in the data outputted by Kilosort, the other main objective of Phy is to assign labels to each cluster, which will be used for later analysis. In Figure 21, an example of ClusterView after manual labelling of all clusters can be observed. Most of the clusters in this example have been classified as Noise. This does not have to be a ground rule, but it is a likely outcome considering the amount of noise that human single neuron recordings are exposed to, even after pre-processing and filtering. Among Noise clusters



there can also be found clusters with few spikes and undetermined clusters which are not considered clean enough to be labelled as anything else. Clusters in green are classified as Good neurons are the main goal of this method. As can be seen, however, the number of neuron clusters is relatively small (only 3 clusters) for an array of 72 clusters. It is a common result for the type of data that was dealt with in this project but can vary according to the characteristics of the recordings. The other two labels are MUA and Unsorted. MUA refers to clusters containing spikes from more than one neuron, but which cannot be effectively separated. In practice, MUA was additionally used as a label to classify clusters in which only some views showed signs of a possible neuron, while the others opposed the possibility. Unsorted, on the other hand, is the default state of clusters before labelling, so the objective is to classify all Unsorted clusters. Experimentally, some clusters can effectively be left as Unsorted to be considered in the following analysis, since it will be valued as a fourth label in the output data. It can be suitable for clusters which are too complicated or ambiguous to classify, with the notion that they could prove useful in post-processing when more specific and power tools are employed.

filter						
id	ch	sh	depth	fr	amp	n_spikes ▼
55	15	0	16.00	16.36	2.91	294477
80	15	0	16.00	14.88	2.41	267767
8	11	0	12.00	5.02	4.70	90438
46	15	0	16.00	3.46	2.44	62234
53	10	0	11.00	2.24	1.80	40357
22	11	0	12.00	2.10	2.75	37719
40	11	0	12.00	2.07	2.59	37326
69	15	0	16.00	1.99	2.52	35888
73	10	0	11.00	1.96	2.07	35323
42	15	0	16.00	1.60	2.66	28735
39	15	0	16.00	1.40	2.19	25169
48	15	0	16.00	1.33	2.68	23923
14	11	0	12.00	1.22	2.37	22024
58	14	0	15.00	1.12	2.52	20158
67	11	0	12.00	0.97	2.47	17537
33	9	0	10.00	0.95	1.99	17164
65	11	0	12.00	0.94	2.01	16835
15	10	0	11.00	0.88	2.17	15769
60	14	0	15.00	0.78	2.38	13953
11	11	0	12.00	0.71	2.30	12709
37	11	0	12.00	0.65	2.43	11714
44	11	0	12.00	0.64	2.11	11530
25	14	0	15.00	0.61	2.30	11002
51	11	0	12.00	0.55	2.26	9936
38	11	0	12.00	0.49	2.12	8860
23	11	0	12.00	0.48	2.32	8592
17	11	0	12.00	0.46	2.75	8354
32	11	0	12.00	0.46	1.97	8257
59	11	0	12.00	0.43	2.07	7778
62	11	0	12.00	0.40	2.29	7226
50	11	0	12.00	0.39	2.02	7092
30	11	0	12.00	0.39	5.44	6933
43	14	0	15.00	0.38	2.56	6791
61	10	0	11.00	0.29	2.05	5304
20	9	0	10.00	0.24	1.70	4389
6	11	0	12.00	0.24	5.01	4249
9	11	0	12.00	0.19	2.75	3361
34	10	0	11.00	0.14	1.74	2546
31	9	0	10.00	0.14	1.55	2538

clusters: 72

Figure 21. ClusterView visualisation after all clusters are manually curated and labelled.

Ultimately, the files obtained in Phy are added to the output of Kilosort, as shown in Figure 22, which has also been modified to adequate the clusters to the new classification realized in Phy. For the post-processing analysis, the most important information that is needed are the spike times and their classification in each cluster and label. This information is saved in three files:

- “spike\_times.npy”: list of spike time of all spikes in samples. It must be divided by the sampling frequency to obtain timestamps of the spikes.
- “spike\_clusters.npy”: list of assigned clusters of all spikes.
- “cluster\_group.tsv”: table of all clusters with the assigned label.

As can be noticed, two of these files are in a NPY format, which needs the npy-matlab toolbox to be generated by Kilosort in Matlab but can be easily read in Python using the Numpy library.

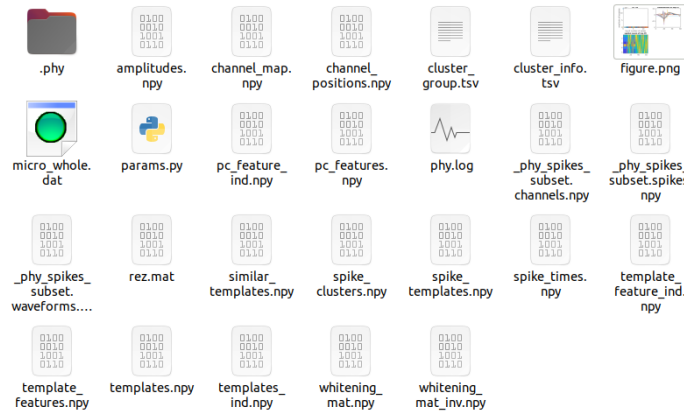
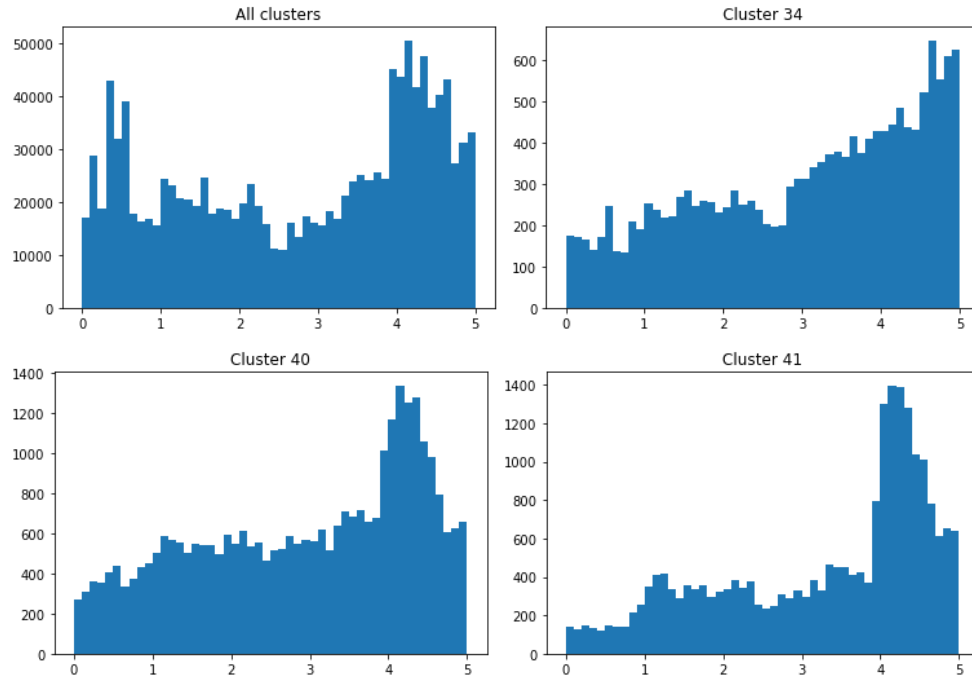


Figure 22. All files produced by Kilosort and Phy.

## Subsequent analysis

The results of Phy can be opened and read in Python with the Numpy library. Data on spikes timestamps, label and sorting cluster can be used in research to obtain information about the firing patterns of neurons. In Figure 23, the spike times of all good neurons detected in a recording, belonging to clusters 34, 40, 41, 47, 65 and 78 as compiled by Kilosort and Phy, are represented using histograms, along with a histogram of all spikes included in Good, MUA, Noise and Unsorted clusters.



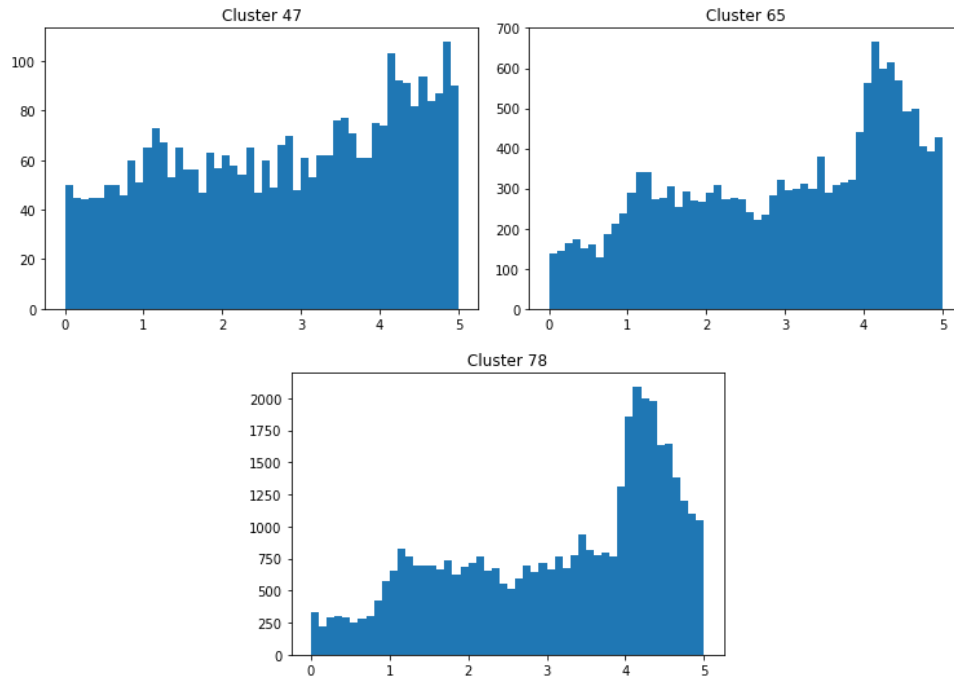


Figure 23. Histograms representing all spike times for all clusters and clusters 34, 40, 41, 47, 65, 78 respectively for the duration of a night of sleep.

Additionally, the firing times of each neuron can be observed plotting the spike times using “matplotlib”, specifically the “pyplot.eventplot” function, which results in a plot like in Figure 24. In this way, firing patterns can be compared between different neurons to interpret the results.

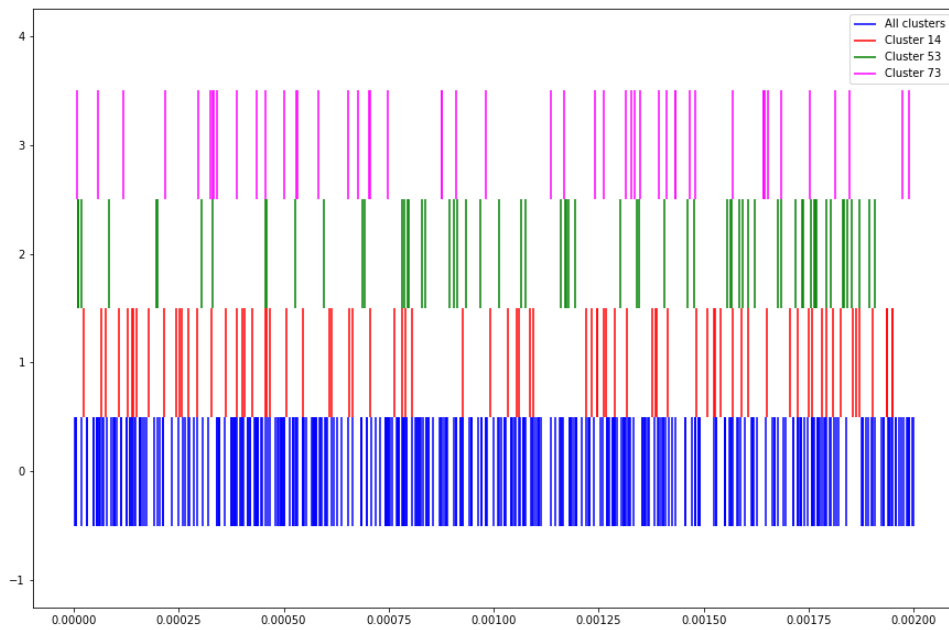


Figure 24. Event plot presenting all spikes during a short period of time of all clusters and clusters 14, 53 and 73.

## 4. Discussion

### Project objective and results overview

The initial objective of the project was to develop a complete pipeline of software applications to analyse single neuron recordings from raw data up to the spike sorting results, linking an array of different methods. After developing and evaluating each of the required methods and being able to implement them in the scientific question posed at the beginning, it can be determined that resulting software has been proved to be successful in managing the data from its raw source to the eventual spike sorting intended. About the methods, it is worth noting that only the pre-processing and filtering was fully developed for the project (except the toolboxes employed). The rest of the methods involved included already developed software and applications. In these cases, the scope of the research was directed towards identifying the best parameter values and developing an application strategy, in order to adapt such pre-established software to the specific requirements of the data.

The pre-processing developed in Matlab has been successful in managing to extract human single neuron data from raw recordings and preparing it for later analysis. Considering the versatility necessary for this kind of method, which is aimed to generalise the method for data importing and standardize its structure, the results have been optimal. With an interface easy to understand and accessible for external developers, it is designed so that the whole script can adapt to new types of data thanks to the powerful Fieldtrip toolbox. More importantly, it is completely scalable and allows for merging of multiple recordings in both the spatial and temporal dimensions. Aside from more efficient memory managing methods, the main limitation of the Matlab script resides in the filter. Due to the long filtering times of this process, a reiterative algorithm that automatically calculates the parameters necessary to apply the notch filter for every frequency is fundamental to accelerate the procedure and improve performance. However, the algorithm developed, based on statistical calculations on the data, is far from perfect. Using spectral as well as additional statistical techniques to study the behaviour of the filter, this method could be improved to more adequately respond to specific characteristics of the data. The main reason why such study has not been implemented, aside from lack of time, is the necessary linkage between this step and all consecutive events of the project, which entails that any far-reaching effects on the data need to be evaluated at several stages of the software. Therefore, such in-depth study would considerably increment the magnitude of the project.

Kilosort has been the main challenge in the matter of software research and adaptation of priorly developed methods. The initial objective to reuse already standardized techniques such as Kilosort of Phy, which had been used in research for years, was severely hampered by the characteristics of human single neuron recordings. The type of data obtained was extensively different from typical mouse or rat recordings, and as such the general methods had to be adapted to them. It has not only been a matter of modifying analytical parameters to obtain better results, but of understanding the complexities of the software (and its different versions) in order to comprehend how our data would respond to certain features. Specially at initial steps, the goal was not to improve the output of the sorting, but to obtain any kind of understandable sorting at all. By cause of extensive research on any modification being designed for the software in the last several years, as well as direct

contact with developers, a strategy that optimizes the resources of Kilosort was formulated in order to be able to process human single neuron signals.

Finally, the last method was the least time expensive, but also the one that required more previous research. In this case, it has been easier to adapt already established methods, since the results of the spike sorting in Kilosort are homogenised for every type of data. However, Phy is such an expansive tool, with so many resources for visualization and manipulation, that a step-by-step guide is needed to make sense of the data and be able to analyse different datasets systematically. The exposed manual aims to simplify a series of processes explained by researchers and developers online, as well as by field specialists, while adjusting the procedure for certain characteristics of our type of data. In the end, it is important to note the manually refining the results of a powerful software such as Kilosort is not trivial, and many features need to be considered at the same time. For this reason, I consider that the best predictor of an optimized application of Phy is experience: the ability to discern details or patterns in the data's characteristics which are difficult to be described experimentally, as well as the knowledge of prior examples of the same situation. Hence, this manual also intends to provide examples and practical cases in order to offer a larger overview of what can be considered good practices in manual refining of spike sorting.

## **Limitations and future development**

This project was developed from limited existent software and tackled a scientific demand that is just now starting to be put forward. For this reason, some steps along the way are mostly experimental and would benefit from in-depth studies of their behaviour to improve performance and overall results. An effective applicability of this project that would lead to further development remains to be seen, but certainly there is plenty of room for improvement.

Nevertheless, there is one specific development that would significantly enhance the chances of this software being used in future research: the unification of the pipeline in a single software, capable of running the whole analysis. A significant portion of the time spend right now in the execution of the software is devoted to transferring data from one system to another for each specific method. This could be avoided by developing all necessary software in the same environment and unifying the code, so as to not have to access intermediate processes' output. Additionally, the simplification of the procedure would make future software developments easier, since whole-pipeline performance could be evaluated much faster. Especially parameters present in the initial steps, which would be benefited by being able to faster and more accurately predict their impact of later results.

However, taking into account that the different methods arise from completely independent environments and have a variety of requirement, such unification is highly improbable unless the whole software is developed from scratch. Alternatively, considering that the pre-processing and Kilosort are both run on Matlab, at least these two methods could be combined in a single pipeline, which would only need to be executed once to obtain directly the spikes sorted from raw data. This merging of software was not undertaken in this project due to time limitation and the fact that their different system requirements made it impossible to run them simultaneously in the same device.

More precise limitations in specific steps of the process can also lead to future development in order to increase efficiency. For instance, one of the most troublesome aspects of the data are the artifacts, results of noise, disconnections from the device or medical conditions. During the pre-processing, it has been hypothesized that artifacts could be identified in the data using whole-brain imaging techniques (where movement or other alterations could be observed), and then removed alongside the notch frequencies in the filter (or using specific Fieldtrip functions). This would lead to cleaner data and theoretically to better sensitivity in the spike sorting step, since it would remove a lot of noise. However, as mentioned above, many of such artifacts are directly removed during the automatic spike sorting and the remaining clusters of noise activity can be relatively easily discarded in the manual refining, so that the classification improvements derived from their removal are debatable and possibly insignificant. Another way to improve the pre-processing and filtering methods would be by testing them with other registrations and even for different types of data, to ensure that the established parameters stand or, alternatively, whether they should be defined manually before each new dataset.

Additionally, the pre-processing and filtering method is a high-demanding computational process which must be met with specific system requirements. Apart from the CUDA requirement, which could be avoided but is highly recommended due to the significantly shortening of waiting times, the RAM available memory necessary to execute the script, above 100 GB to be run smoothly, cannot be matched by most conventional personal computers. Moreover, even in research environments where such systems are at use, this large occupation of memory can be detrimental to other processes and represent an inconvenience. Due to time limitation and the readily availability of large memory systems in our working environment, an adaptation of the software for less favourable systems was deemed unnecessary. However, there are many methods natively available in Matlab to optimise memory consumption, as well as external toolboxes, that could be implemented to reduce this requirement. Similarly, it is probable that a deep study and restructuring of the data organization could lead to more efficient memory usage.

Regarding the notch filter, necessary for the cleaning of data before being inputted to Kilosort, could see significant improvement and future development with better studying its features and inner working. The filter parameters proposed in this project are an approach to make the filter more intelligent and flexible, mainly from an experimental point of view. Specifically studying the behaviours of the filter and its parameters could lead to finer tuning of results and even improve the following methods, in particular the automatic Kilosort spike sorting. Furthermore, such study would be greatly aided by the unification of the software in a single process, as explained before, which would accelerate its various analysis and thus offer more detailed results in shorter times.

Similar to the filter approach, the main parameter in Kilosort which affects sorting results is the detection threshold. In this case, a superficial study was carried out so as to discern which was broadly the best combination of values for the extraction of spikes in our data. However, an in-depth analysis of the impact of this parameter, especially in diverse settings and with different inputs, would provide more accurate analytical power and probably better sorting results. Moreover, many aspects of the Kilosort and Phy applications are implemented experimentally, after many layers of adaptation and trial and error. In such cases, experience and further development could only improve their performance and optimize the procedure.

## Economic study and budget

Due to the heavy processing needs of some functions, the system used to implement the software should have two main characteristics: it must possess a graphics processing unit (GPU), such as CUDA NVIDIA, in order to enable high-performance computationally intensive operations (Baig et al., 2020), and it must include at least around 50 gigabytes of RAM memory.

Note that both these characteristics are not truly requirements, as the software could be implemented in more simple environments as well, however then the most computationally demanding processes would take exceptionally long times to be carried out (for more basic systems, it could take weeks to execute a given task in Matlab or Kilosort). Additionally, these requirements are needed separately for independent tasks, so they can be split between different systems with different computational capabilities. Specifically, data preprocessing and filtering requires large RAM memory due to the parallel allocation of significant amounts of data, whilst Kilosort is very computationally demanding, so a GPU is needed to greatly reduce execution times. In our case, for instance, all Matlab processes were run in a centralized server with over 500 gigabytes of RAM memory and the rest of the pipeline was developed in a CUDA-enabled portable computer. Nvidia offers a wide range of GPUs to choose, each with different capabilities. On this project a GeForce model was employed, which have a price range of 439 to 2249€.

Similar to the hardware requirements, software is also divided among different applications, each of which implies a fixed cost in terms of licenses. Of the three main software applications needed, only Matlab requires a license, since Kilosort and Phy both are open-source software. Matlab licenses can range from 35 to 800€ in annual licenses, and up to 2000€ in perpetual licenses. Moreover, during the filtering the DSP System Toolbox is employed, which is linked to Matlab but needs to be purchased separately. The DSP System Toolbox has a cost of 500€ (annual) or 1250€ (perpetual).

Apart from the basic costs of hardware and software, the cost of development must be taken into account, including coding and evaluation times. It is calculated that the total time spent on the writing of the code, analysis, evaluation and implementation for use amounts to roughly 200 hours. However, in case the software was meant to be exported for commercial purposes, additional time should be invested in documentation preparation and further adaptation of the code to a user-friendly interface. Considering only the current time expenditure, and an average salary rate of around 12€ per hour, the total development cost is equal to 2400€.

Overall, the entire cost of the project, including hardware, software and developmental costs, would range from 3374 to 7899€. Considering only the software and development, it would have an average cost of around 4300€. In case a software package was assembled distributed as a product, it could include all licenses to the software plus the development cost, or only the additional software that is implemented on Matlab. Leaving the Matlab license out, the cost of the product could be reduced to 3300€ on average.

## Confidential aspects

The entirety of the software employed for the project is open source and public, so its usage did not require any type of confidentiality accord, since no exchange of information was needed to access the software. Furthermore, we had limited contact with other companies or research groups, so no discussion of sensitive information was involved.

During the development of the project and specially in testing, data from human single neuron recordings were employed. The data was provided by health institutions and mainly contained clinical information. As such, the topic of medical confidentiality is relevant to this question.

Privacy and confidentiality are fundamental rights and have to be particularly looked after and protected in research settings. Participants in any kind of research must have the right to allow the use of their personal data and have control over it for the entirety of the study's duration. It is considered an ethical duty of researchers to protect patient information restricting unlawful access, modification, usage, disclosure or loss. The Canadian Institutes for Health Research (CIHR), the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Social Sciences and Humanities Research Council (SSHRC) have suggested a series of categories defining which type of data could be used to identify a research participant and thus deserves more protection (Harnett, 2021):

- “Directly identifying information—the information identifies a specific individual through direct identifiers (e.g., name, social insurance number, personal health number).
- Indirectly identifying information—the information can reasonably be expected to identify an individual through a combination of indirect identifiers (e.g., date of birth, place of residence or unique personal characteristic).
- Coded information—direct identifiers are removed from the information and replaced with a code. Depending on access to the code, it may be possible to re-identify specific participants (e.g., the principal investigator retains a list that links the participants' code names with their actual names, so data can be re-linked if necessary).
- Anonymized information—the information is irrevocably stripped of direct identifiers, a code is not kept to allow future re-linkage, and risk of re-identification of individuals from remaining indirect identifiers is low or very low.
- Anonymous information—the information never had identifiers associated with it (e.g., anonymous surveys) and risk of identification of individuals is low or very low”

According to these categories, the type of data that we use can be classified as Anonymized information, due to the fact that at least from the point of view of our connection to the data, it is not linked to direct or indirect identifiers that could lead to the individual's identity. It remains to be seen the kind of information about the data that is kept by the providers, to study the actual privacy risk in case of tracing back the data (Harnett, 2021).



## Execution chronogram

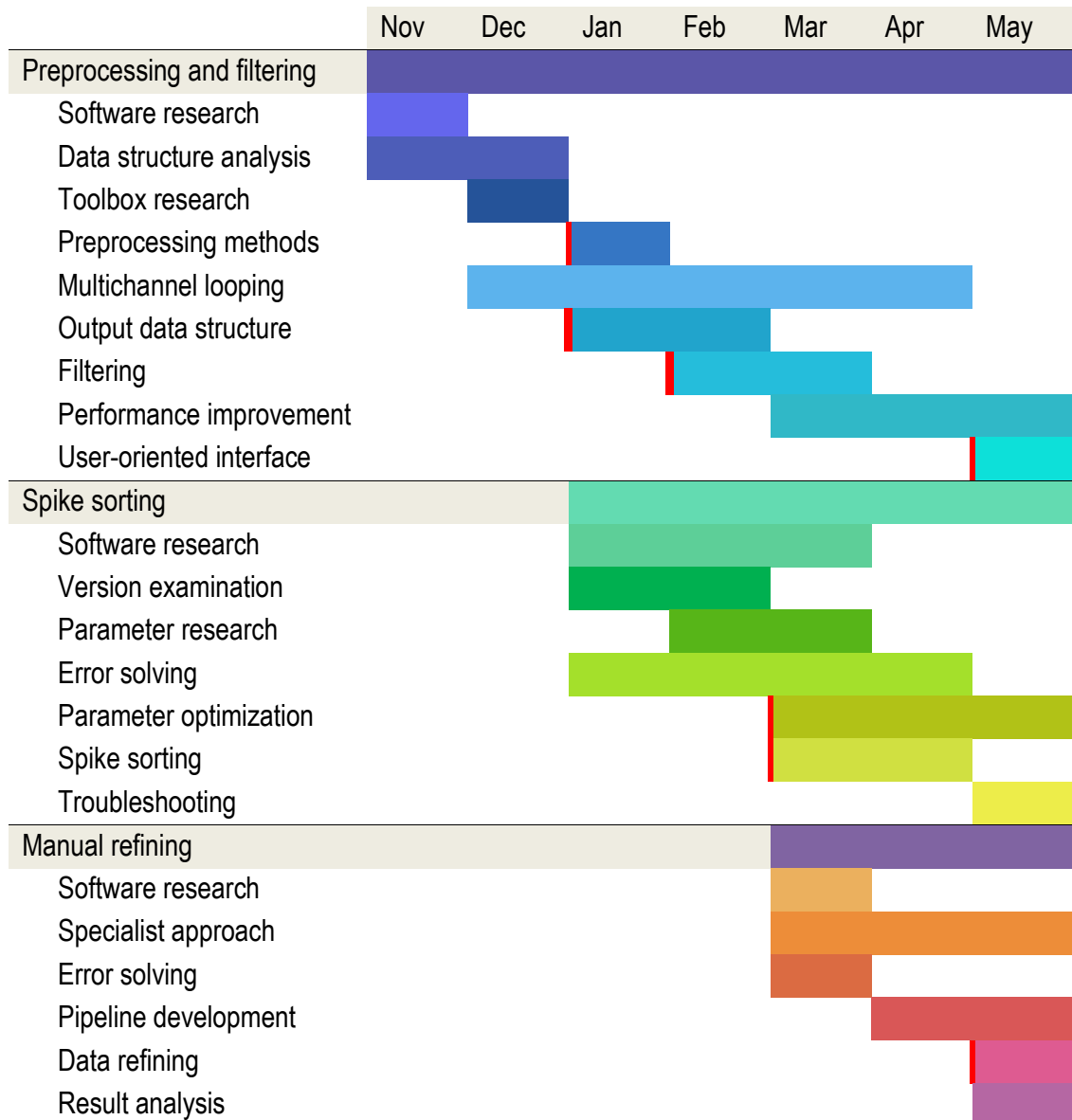


Figure 25. Execution chronogram of the entire project, subdividing the three methods into main tasks including all steps involved in the development and testing of the software.

The main aspect to consider is the fact that due to the lineal behaviour of the whole software, also its different methods were developed according to a workflow in which the results of one method were necessary to make progress in the next one. Using a sample dataset to write the code around it also made this type of lineal development necessary, since only data that had already been through the prior method could be applied to the following one. At the same time, however, most of the time spent in the project revolved around researching possible software and the behaviour of their parameters, as well as error solving and troubleshooting, which limited the progress of the workflow due to the need to correct such issues before being able to advance. Such research was often undertaken parallel to other processes, so that many steps overlap.

Processes that were inherently limited by prior developments and had to wait for results from previous methods are marked with a red line. Among this, there can be found the spike sorting and the manual refining effectuated on the test data, to evaluate the performance of the methods. These steps necessarily relied on data that had already undergone the previous stages of the procedure, so until the figuring out of the output data structure and the spikes sorting were effectuated, respectively, they could not be started. Regarding the Matlab code, other steps that could not be started until the end of previous processes include the pre-processing methods and defining the output data structure, since both of them depended on the initial data structure analysis and toolbox research. Similarly, filtering could not be started until the pre-processing of the data was undertaken. Also pre-processing and filtering, the user-oriented interface could not be developed priorly to the multichannel looping, since this is the base code that was modified. Multichannel looping and user-oriented interface are directly related, because user-oriented interface also includes looping and automatic algorithms to make the process easier. However, the first is meant to increase efficiency and speed, while the latter is meant to make the user experience more accessible. Finally, in spike sorting the parameter optimization required to have decided on a specific software version and an initial parameter research.

Overall, the later a method was needed for the whole procedure, the later it was begun to be developed, and in every case each method started with an in-depth software research of the possible options available nowadays and the most suitable approach for the specificities of our data analysis, for this reason each research included different mechanism. For instance, the software research in Kilosort included consulting developers and well as community discussions on experimental results. Additionally, it is worth noting that all the methods have been developed until the end, because modifications at later stages revealed new aspects in which prior steps could be improved to optimise the results, so a constant activity of adaptation was undertaken.

## **SWOT analysis**

The main objective of the project was to develop a research tool that could be implemented in human single neuron studies. As such, it was developed in a research environment with specialists in the brain imaging and neuroscience fields. In case of this project being used in future research or further developed, in this section a SWOT analysis is presented.

A SWOT analysis is a strategic planning technique for assessing four key values of a project: the Strength, Weaknesses, Opportunities and Threats. Strengths and Weaknesses refer to the intrinsic characteristics of the project, while Opportunities and Threats represent the position that the project occupies in a specific environment and its relationship with other projects.

- **Strengths**

During the development of the software, many different ways to approach the process were tackled, denoting a high degree of specialisation to the type of data, since most parallel paths were unfeasible or led to poor results.

Additionally, considering the focus on parameter study and optimization, a pipeline for quality enhancement based on better adjustment of analytic values would be easy to develop. This way, the project has a large capacity for growth in future instances.

- **Weaknesses**

Working with external software, instead of an entirely self-developed code, implies that some of the features required to consider for the analysis might not be fully compliant with the needs of our project or our data. For this reason, communication with and dependability on developers has been a key aspect in the understanding and improvement of software capabilities, which has involved an additional expenditure of time and resources. Moreover, such expenditure might be further needed if future development of the project is ever fostered.

- **Opportunities**

A unification of methods as contemplated priorly, combining all software into the same system, would increase efficiency and reduce the significant processing times, making it more competitive than any alternative software. Alternatively, only considering the pre-processing and filtering and the spike sorting methods being merged in a unified package fully developed in Matlab would already prove highly beneficial. Similarly, any post-processing analysis could be paired with Phy to simplify all methods developed in a Python environment.

The main advantage that would arise from such configuration would not only be the reduction in preparation time, since the current assembling demands the data to be moved between systems after every method, but also remove possible errors emerging from the loss or mismanagement of data during the migration process.

On the other hand, human single neuron being an emerging technology, it could see its range of application significantly broadened in the near future.

- **Threats**

As it is not a compact software but a juxtaposition of analytical steps, any external unified package that could arise, providing the same methods in an integrated interface, would be remarkably more convenient and thus it would obviate the need for the present software.

This fact is reinforced by the need for independent systems in the execution of different steps and the high requirement needs of some methods.

Additionally, the lack of research on human single neuron signals makes it an uncertain technology with limited predictability. Future developments could make this technology obsolete or simply unnecessary, as a result of improvement of alternative technology as much as abandonment of this research field.

## 5. Bibliography

- Baig, F., Gao, C., Teng, D., Kong, J., & Wang, F. (2020). Accelerating Spatial Cross-Matching on CPU-GPU Hybrid Platform With CUDA and OpenACC. *Frontiers in Big Data*, 3. <https://doi.org/10.3389/fdata.2020.00014>
- Borbély, A. A., Daan, S., Wirz-Justice, A., & Deboer, T. (2016). The two-process model of sleep regulation: a reappraisal. *Journal of Sleep Research*, 25(2), 131–143. <https://doi.org/10.1111/jsr.12371>
- Dalmau, J., Gleichman, A. J., Hughes, E. G., Rossi, J. E., Peng, X., Lai, M., Dessain, S. K., Rosenfeld, M. R., Balice-Gordon, R., & Lynch, D. R. (2008). Anti-NMDA-receptor encephalitis: case series and analysis of the effects of antibodies. *The Lancet Neurology*, 7(12), 1091–1098. [https://doi.org/10.1016/S1474-4422\(08\)70224-2](https://doi.org/10.1016/S1474-4422(08)70224-2)
- Diekelmann, S., & Born, J. (2010). The memory function of sleep. *Nature Reviews Neuroscience*, 11(2), 114–126. <https://doi.org/10.1038/nrn2762>
- DSP System Toolbox Documentation. (n.d.). Retrieved May 7, 2022, from <https://www.mathworks.com/products/dsp-system.html>
- Gunduz-Bruce, H. (2009). The acute effects of NMDA antagonism: From the rodent to the human brain. *Brain Research Reviews*, 60(2), 279–286. <https://doi.org/10.1016/j.brainresrev.2008.07.006>
- Harnett, J. D. (2021). *Research Ethics for Clinical Researchers* (pp. 53–64). [https://doi.org/10.1007/978-1-0716-1138-8\\_4](https://doi.org/10.1007/978-1-0716-1138-8_4)
- Hughes, E. G., Peng, X., Gleichman, A. J., Lai, M., Zhou, L., Tsou, R., Parsons, T. D., Lynch, D. R., Dalmau, J., & Balice-Gordon, R. J. (2010). Cellular and synaptic mechanisms of anti-NMDA receptor encephalitis. *The Journal of Neuroscience : The Official Journal of the Society for Neuroscience*, 30(17), 5866–5875. <https://doi.org/10.1523/JNEUROSCI.0167-10.2010>
- Ishiura, H., Matsuda, S., Higashihara, M., Hasegawa, M., Hida, A., Hanajima, R., Yamamoto, T., Shimizu, J., Dalmau, J., & Tsuji, S. (2008). RESPONSE OF ANTI-NMDA RECEPTOR ENCEPHALITIS WITHOUT TUMOR TO IMMUNOTHERAPY INCLUDING RITUXIMAB. *Neurology*, 71(23), 1921–1923. <https://doi.org/10.1212/01.wnl.0000336648.43562.59>
- Kanthida van Welzen. (n.d.). *Decreased overnight slow wave slope change in anti-NMDAR encephalitis and schizophrenia*.
- Kilosort Documentation. (n.d.). Retrieved May 15, 2022, from <https://github.com/cortex-lab/KiloSort>
- Kubska, Z. R., & Kamiński, J. (2021). How Human Single-Neuron Recordings Can Help Us Understand Cognition: Insights from Memory Studies. *Brain Sciences*, 11(4), 443. <https://doi.org/10.3390/brainsci11040443>

- Lau, C. G., & Zukin, R. S. (2007). NMDA receptor trafficking in synaptic plasticity and neuropsychiatric disorders. *Nature Reviews Neuroscience*, 8(6), 413–426. <https://doi.org/10.1038/nrn2153>
- Marius Pachitariu, Nick Steinmetz, Shabnam Kadir, Matteo Carandini, & Jenet Harris. (2016). Fast and accurate spike sorting of high-channel count probes with KiloSort. *Advances in Neural Information Processing Systems* 29.
- Matlab Documentation. (n.d.).
- Misra, A., Burke, J. F., Ramayya, A. G., Jacobs, J., Sperling, M. R., Moxon, K. A., Kahana, M. J., Evans, J. J., & Sharan, A. D. (2014). Methods for implantation of micro-wire bundles and optimization of single/multi-unit recordings from human mesial temporal lobe. *Journal of Neural Engineering*, 11(2), 026013. <https://doi.org/10.1088/1741-2560/11/2/026013>
- Oostenveld, R., Fries, P., Maris, E., & Schoffelen, J.-M. (2011). FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data. *Computational Intelligence and Neuroscience*, 2011, 1–9. <https://doi.org/10.1155/2011/156869>
- Phy Documentation. (n.d.). Retrieved May 15, 2022, from <https://github.com/cortex-lab/phy>
- Quiroga, R. Q., Reddy, L., Kreiman, G., Koch, C., & Fried, I. (2005). Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045), 1102–1107. <https://doi.org/10.1038/nature03687>
- Riedner, B. A., Vyazovskiy, V. v., Huber, R., Massimini, M., Esser, S., Murphy, M., & Tononi, G. (2007). Sleep Homeostasis and Cortical Synchronization: III. A High-Density EEG Study of Sleep Slow Waves in Humans. *Sleep*, 30(12), 1643–1657. <https://doi.org/10.1093/sleep/30.12.1643>
- Shepherd, J. D., & Huganir, R. L. (2007). The Cell Biology of Synaptic Plasticity: AMPA Receptor Trafficking. *Annual Review of Cell and Developmental Biology*, 23(1), 613–643. <https://doi.org/10.1146/annurev.cellbio.23.090506.123516>
- Tononi, G., & Cirelli, C. (2014). Sleep and the Price of Plasticity: From Synaptic and Cellular Homeostasis to Memory Consolidation and Integration. *Neuron*, 81(1), 12–34. <https://doi.org/10.1016/j.neuron.2013.12.025>
- Vyazovskiy, V. v., Olcese, U., Lazimy, Y. M., Faraguna, U., Esser, S. K., Williams, J. C., Cirelli, C., & Tononi, G. (2009). Cortical Firing and Sleep Homeostasis. *Neuron*, 63(6), 865–878. <https://doi.org/10.1016/j.neuron.2009.08.024>