SpyKING CIRCUS Documentation

Release 0.5.0

Pierre Yger and Olivier Marre

Jul 28, 2017

Contents

The SpyKING CIRCUS is a massively parallel code to perform semi automatic spike sorting on large extra-cellular recordings. Using a smart clustering and a greedy template matching approach, the code can solve the problem of overlapping spikes, and has been tested both for *in vitro* and *in vivo* data, from tens of channels to up to 4225 channels. Results are very good, cross-validated on several datasets, and details of the algorithm can be found in the following publication: http://biorxiv.org/content/early/2016/08/04/067843

Note: In 0.5, smart search is now activated by default, and important changes have been done under the hood to refactor the code in order to read/write virtually any file format, as long as a wrapper is provided.

- Be in touch if you are interested by writing your own file wrapper to read/write a custom file format.
- A "garbage collector" mode has also been added, to help the user to get a **qualitative** feedback on the performance of the algorithm, estimating the number of missed spikes: if `collect_all` in the `[fitting]` section is activated, spikes left unfitted during the fitting procedure can be collected, grouped by electrode (keep in mind that this is more for a debugging purpose, and those spikes are only an approximation).
- A `smart_select` option is now available in the `[clustering]` section, that should considerably enhance the clustering for datasets with very low or very high firing rates. If feedback and some more quantitative benchmarks are positive, it may become the new default mode for clustering. You are welcome to give it a try

---

CHAPTER 1

Introduction

In this section, you will find all basic information you need about the software. Why you should use it or at least give it a try, how to get it, and how to install it. To know more about how to use it, see the following sections.

Why using it?

SpyKING CIRCUS is a free, open-source, spike sorting software written entirely in python. In a nutshell, this is a fast and efficient way to perform spike sorting using a template-matching based algorithm.

Because you have too many channels

Classical algorithms of spike sorting are not properly scaling up when the number of channels is increasing. Most, if not all of them would have a very hard time dealing with more than 100 channels. However, the new generation of electrodes, either *in vitro* (MEA with 4225 channels) or *in vivo* (IMEC probe with 128 channels) are providing more and more channels, such that there is a clear need for a software that would properly scale with the size of the electrodes.

Note: → The SpyKING CIRCUS, based on the MPI library, can be launched on several processors. Execution time scales linearly as function of the number of computing nodes, and the memory consumption scales only linearly as function of the number of channels. So far, the code can handle 4225 channels in parallel.

Because of overlapping spikes

With classical spike sorting algorithms, overlapping spikes are leading to outliers in your clusters, such that they are discarded. Therefore, each time two neurons have overlapping waveforms, their spikes are ignored. This can be problematic when you are addressing questions relying on fine temporal interactions within neurons. It is even more problematic with large and dense electrodes, with many recording sites close from each others, because those

overlapping spikes start to be the rule instead of the exception. Therefore, you need to have a spike sorting algorithm that can disentangle those overlapping spikes.

Note: → The SpyKING CIRCUS, using a template-matching based algorithm, reconstructs the signal as a linear sum of individual waveforms, such that it can resolve the fine cross-correlations between neurons.

Because you want to automatize

For large number of channels, a lot of clusters (or equivalently templates, or cells) can be detected by spike sorting algorithms, and the time spent by a human to review those results should be reduced as much as possible.

Note: → The SpyKING CIRCUS, in its current form, aims at automatizing as much as possible the whole workflow of spike sorting, reducing the human interaction. Not that it can be zero, but the software aims toward a drastic reduction of the manual curation, and results shows that performances as good or even better than with classical spike sorting approaches can be obtained (obtaining for example one of the best score on synthetic data).

How to get the code

The code is currently hosted on github, in a public repository, relying on Git, at https://github.com/spyking-circus/spyking-circus. The following explanations are only for those that want to get a copy of the git folder, with a cutting-edge version of the software.

Note: The code can be installed automatically to its latest release using `pip` or `conda` (see *How to install*).

Cloning the source

Create a folder called `spyking-circus`, and simply do:

```
>> git clone https://github.com/spyking-circus/spyking-circus.git spyking-circus
```

The advantages of that is that you can simply update the code, if changes have been made, by doing:

```
>> git pull
```

Without git

If you do not have git installed, and want to get the source, then one way to proceed is:

1. Download and install SourceTree 2. 3. Click on the `Clone in SourceTree` button, and use the following link with SourceTree https://github.com/spyking-circus/spyking-circus 4. In SourceTree you just need to click on the `Pull` button to get the latest version of the software.

For Windows users

Note: See the install section for Windows user here

For Mac OS X users

In order to install MPI, you may need to download the Xcode tools

Note: See the install section for Mac user here

Download the archive

All released versions of the code can now be downloaded in the `Download` section of the github project, as `.tar.gz` files (pip install)

To know more about how to install the sofware, (see *How to install*)

Installation

The SpyKING CIRCUS comes as a python package, and it at this stage, note that mostly unix systems have been tested. However, users managed to get the software running on Mac OS X, and on Windows 7,8, or 10. We are doing our best, using your feedbacks, to improve the packaging and make the whole process as smooth as possible on all platforms.

How to install

Note: If you are a Windows or a Mac user, we recommend using Anaconda, and:

- see here for detailed instructions on Windows
- see here for detailed instructions on Mac OS X

Using with CONDA

Install Anaconda or miniconda, e.g. all on the terminal (but there is also a .exe installer for Windows, etc.):

As an example for linux, just type:

```
>> wget https://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh
>> bash Miniconda-latest-Linux-x86_64.sh
```

Then install the software itself:

```
>> conda install -c conda-forge -c spyking-circus spyking-circus
```

If you want to get a support for GPU, see the devoted section on CUDA.

Using pip

To do so, use the `pip` utility:

```
>> pip install spyking-circus
```

You might want to add the `--user` flag, to install SpyKING CIRCUS for the local user only, which means that you don't need administrator privileges for the installation.

In principle, the above command also install SpyKING CIRCUS's dependencies, and CUDA support if `nvcc` command is found in your environment. Once the install is complete, you need to add the `PATH` where SpyKING CIRCUS has been installed into your local `PATH`, if not already the case. To do so, simply edit your `$HOME/.bashrc` and add the following line:

```
export PATH=$PATH:$HOME/.local/bin
```

Then you have to relaunch the shell, and you should now have the SpyKING CIRCUS installed!

Using sources

Alternatively, you can download the source package directly and uncompress it, or work directly with the git folder https://github.com/spyking-circus/spyking-circus to be in sync with bug fixes. You can then simply run:

```
>> pip install . --user
```

Or even better, you can install it with the develop mode:

```
>> pip install . -e --user
```

Such that if you do a git pull in the software directory, you do not need to reinstall it.

For those that are not pip users, it is equivalent to:

```
>> python setup.py install
```

Or to keep the folder in sync with the install in a develop mode:

```
>> python setup.py develop
```

Note: If you want to install `scikit-learn`, needed to get the BEER estimates, you need to add `[beer]` to any pip install

---

Note: If you experience some issues with Qt4 or pyQt, you may need to install it manually on your system. For linux users, simply use your software distribution system (apt for example). For windows user, please see here

---

Activating CUDA

Using CUDA can, depending on your hardware, **drastically** increase the speed of algorithm. However, in 0.5, 1 GPU is faster than 1 CPU but not faster than several CPUs. This is something we are working on to improve in future releases. To use your GPU, you need to have a working CUDA environment installed onto the machine. During the pip installation, the code should automatically detect it and install CUDA bindings if possible. Otherwise, to get support for the GPU with an Anaconda install, just do:

```
>> pip install https://github.com/yger/cudamat/archive/master.zip#egg=cudamat-0.
↪3circus
```

Note: You must have a valid CUDA installation, and `nvcc` installed. If you do not want CUDAMAT to be install automatically, simply do `python setup.py install --nocuda` while installing the software

Home Directory

During the install, the code creates a `spyking-circus` folder in `/home/user` where it will copy several probe designs, and a copy of the default parameter file. Note that if you are always using the code with a similar setup, you can edit this template, as this is the one that will be used by default.

Parallelism

Using MPI

If you are planning to use MPI, the best solution is to create a file `$HOME/spyking-circus/circus.hosts` with the lists of available nodes (see *Configuration of MPI*). You should also make sure, for large number of electrodes, that your MPI implementation is compatible recent enough such that it can allow shared memory within processes.

Using HDF5 with MPI

If you are planning to use large number of electrodes (> 500), then you may use the fact that the code can use parallel HDF5. This will speed everything and reduce disk usage. To know more about how to activate it, see (see Parallel HDF5).

Dependencies

For information, here is the list of all the dependencies required by the SpyKING CIRCUS:

1. `tqdm`
2. `mpi4py`
3. `numpy`
4. `cython`
5. `scipy`
6. `matplotlib`
7. `h5py`
8. `colorama`
9. `cudamat` [optional, CUDA only]
10. `sklearn` [optional, only for BEER estimate]

Configuration of MPI

The code is able to use multiple CPU to speed up the operations. It can even use GPU during the fitting phase. However, you need to have a valid hostfile to inform MPI of what are the available nodes on your computer. By default, the code searches for the file `circus.hosts` in the spyking-circus folder, create during the installation `$HOME/spyking-circus/`. Otherwise, you can provide it to the main script with the `-H` argument (see *documentation on the parameters*):

```
>> spyking-circus path/mydata.extesion -H mpi.hosts
```

Structure of the hostfile

Such a hostfile may depend on the fork of MPI you are using. For OpenMPI, this will typically look like:

```
192.168.0.1 max-slots=4
192.168.0.2 max-slots=4
192.168.0.3 max-slots=4
192.168.0.4 max-slots=4
192.168.0.5 max-slots=4
```

If this is your parameter file, and if you launch the code with 20 CPUs:

```
>> spyking-circus path/mydata.extension -c 20
```

Then the code will launch 4 instances of the program on the 5 nodes listed in the hostname.hosts file

---

Note: If you are using multiple machines, all should read/write in a **shared** folder. This can be done with NFS or SAMBA on Windows. Usually, most clusters will provide you such a shared `/home/user` folder, be sure this is the case

---

Shared Memory

With recent versions of MPI, you can share memory on a single machine, and this is used by the code to reduce the memory footprint. If you have large number of channels and/or templates, be sure to use a recent version of MPI (OpenMPI > 1.8.5 for example)

Handling of GPUs

By default, the code will assume that you have only one GPU per nodes. If this is not the case, then you need to specify the number of GPUs and the number of CPUs when launching the code. For example:

```
>> spyking-circus path/mydata.extension -c 5 -g 10
```

This will tell the code that because `n_gpu` is larger than `n_cpu`, several GPUs per nodes must be assumed. In this particular example, 2 GPUs per nodes.

Warning: Currently, clusters with heterogeneous numbers of GPUs per nodes are not properly handled. Be in touch if interested by the functionality

Release notes

Spyking CIRCUS 0.5

This is the 0.5 release of the SpyKING CIRCUS, a new approach to the problem of spike sorting. The code is based on a smart clustering with sub sampling, and a greedy template matching approach, such that it can resolve the problem of overlapping spikes. The publication about the software is available at http://biorxiv.org/content/early/2016/08/04/067843

Warning: The code may still evolve. Even if results are or should be correct, we can expect some more optimizations in a near future, based on feedbacks obtained on multiple datasets. If you spot some problems with the results, please be in touch with pierre.yger@inserm.fr

Contributions

Code and documentation contributions (ordered by the number of commits):

- Pierre Yger
- Marcel Stimberg
- Baptiste Lebfevre
- Christophe Gardella
- Olivier Marre
- Cyrille Rossant

Release 0.5.3

- Improvement in the smart_select option given various datasets
- Fix a regression for the clustering introduced in 0.5.2

Release 0.5.2

- fix for the MATLAB GUI
- smart_select can now be used [experimental]
- fix for non 0: DISPLAY
- cosmetic changes in the clustering plots
- ordering of the channels in the openephys wrapper
- fix for rates in the MATLAB GUI
- artefacts can now be given in ms or in timesteps with the trig_unit parameter

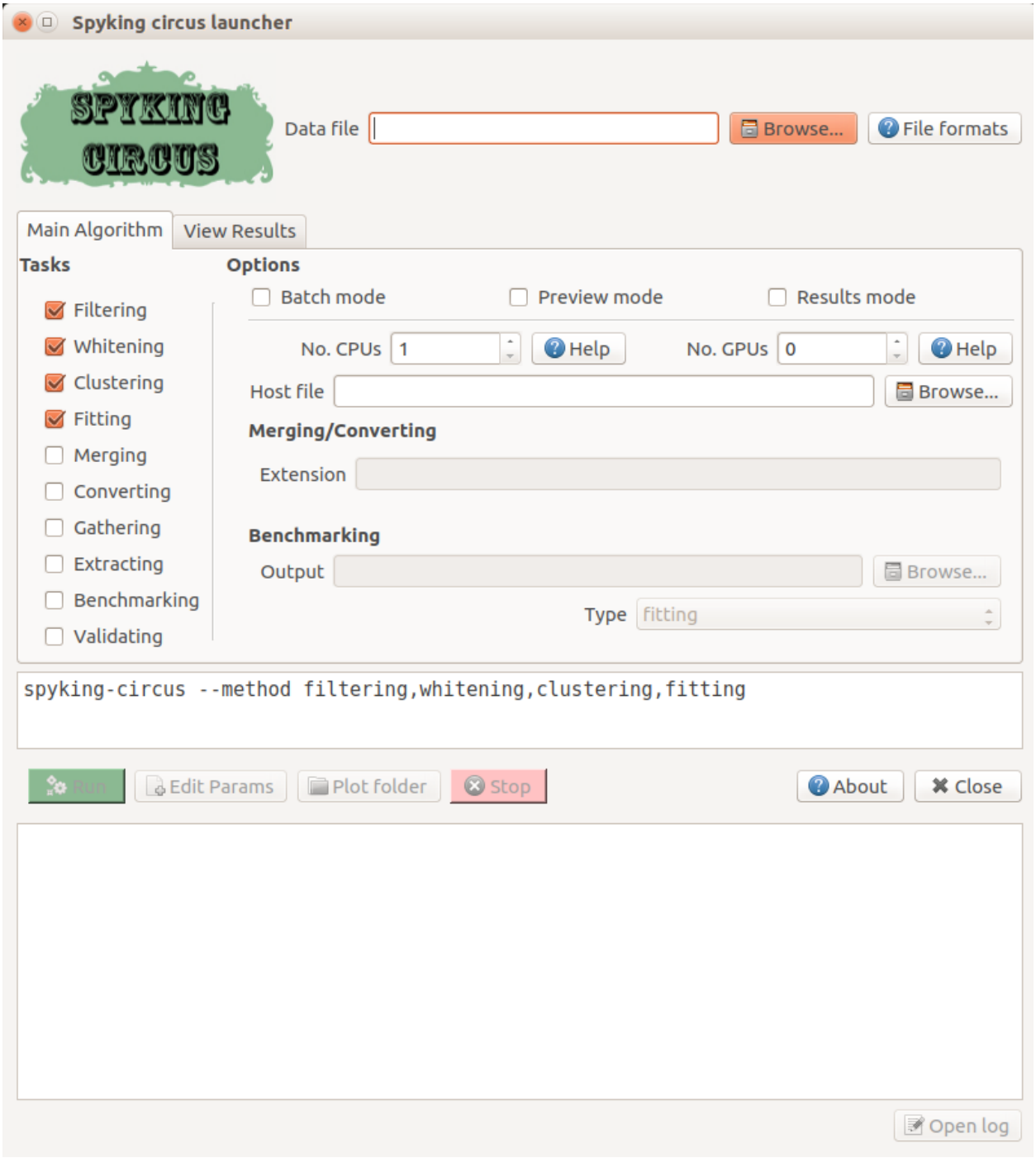


Fig. 1.1: The software can be used with command line, or a dedicated GUI

Release 0.5rc

- fix a bug when exporting for phy in dense mode
- compatibility with numpy 1.12
- fix a regression with artefact removal
- fix a display bug in the thresholds while previewing with a non unitary gain
- fix a bug when filtering in multi-files mode (overwrite False, various t_starts)
- fix a bug when filtering in multi-files mode (overwrite True)
- fix a bug if matlab gui (overwrite False)
- fix the gathering method, not working anymore
- smarter selection of the centroids, leading to more clusters with the smart_select option
- addition of a How to cite section, with listed publications

Release 0.5b9

- switch from progressbar2 to tqdm, for speed and practical issues
- optimization of the ressources by preventing numpy to use multithreading with BLAS
- fix MPI issues appearing sometimes during the fitting procedure
- fix a bug in the preview mode for OpenEphys files
- slightly more robust handling of openephys files, thanks to Ben Acland
- remove the dependency to mpi4py channel on osx, as it was crashing
- fix a bug in circus-multi when using extensions

Release 0.5b8

- fix a bug in the MATLAB GUI in the BestElec while saving
- more consistency with "both" peak detection mode. Twice more waveforms are always collect during whitening/clustering
- sparse export for phy is now available
- addition of a dir_path parameter to be compatible with new phy
- fix a bug in the meta merging GUI when only one template left

Release 0.5b7

- fix a bug while converting data to phy with a non unitary gain
- fix a bug in the merging gui with some version of numpy, forcing ucast
- fix a bug if no spikes are detected while constructing the basis
- Optimization if both positive and negative peaks are detected
- fix a bug with the preview mode, while displaying non float32 data

Release 0.5b6

- fix a bug while launching the MATLAB GUI

Release 0.5b3

- code is now hosted on GitHub
- various cosmetic changes in the terminal
- addition of a garbage collector mode, to collect also all unfitted spikes, per channel
- complete restructuration of the I/O such that the code can now handle multiple file formats
- internal refactoring to ease interaction with new file formats and readibility
- because of the file format, slight restructuration of the parameter files
- N_t and radius have been moved to the [detection] section, more consistent
- addition of an explicit file_format parameter in the [data] section
- every file format may have its own parameters, see documentation for details (or –info)
- can now work natively with open ephys data files (.openephys)
- can now work natively with MCD data files (.mcd) [using neuroshare]
- can now work natively with Kwik (KWD) data files (.kwd)
- can now work natively with NeuroDataWithoutBorders files (.nwb)
- can now work natively with NiX files (.nix)
- can now work natively with any HDF5-like structure data files (.h5)
- can now work natively with Arf data files (.arf)
- can now work natively with 3Brain data files (.brw)
- can now work natively with Numpy arrays (.npy)
- can now work natively with all file format supported by NeuroShare (plexon, blackrock, mcd, ...)
- can still work natively with raw binary files with/without headers :)
- faster IO for raw binary files
- refactoring of the exports during multi-file/preview/benchmark: everything is now handled in raw binary
- fix a bug with the size of the safety time parameter during whitening and clustering
- all the interactions with the parameters are now done in the circus/shared/parser.py file
- all the interactions with the probe are now done in the circus/shared/probes.py file
- all the messages are now handled in circus/shared/messages.py
- more robust and explicit logging system
- more robust checking of the parameters
- display the electrode number in the preview/result GUI
- setting up a continuous integration workflow to test all conda packages with appveyor and travis automatically
- cuda support is now turned off by default, for smoother install procedures (GPU yet do not bring much)

- file format can be streamed. Over several files (former multi-file mode), but also within the same file
- several cosmetic changes in the default parameter file
- clustering:smart_search and merging:correct_lag are now True by default
- fix a minor bug in the smart search, biasing the estimation of densities
- fix a bug with the masks and the smart-search: improving results
- addition of an overwrite parameter. Note that any t_start/t_stop infos are lost
- if using streams, or internal t_start, output times are on the same time axis than the datafile
- more robust parameter checking

Release 0.4.3

- cosmetic changes in the terminal
- suggest to reduce chunk sizes for high density probes (N_e > 500) to save memory
- fix a once-in-a-while bug in the smart-search

Release 0.4.2

- fix a bug in the test suite
- fix a bug in python GUI for non integer thresholds
- fix a bug with output strings in python3
- fix a bug to kill processes in windows from the launcher
- fix graphical issues in the launcher and python3
- colors are now present also in python3
- finer control of the amplitudes with the dispersion parameter
- finer control of the cut off frequencies during the filtering
- the smart search mode is now back, with a simple True/False flag. Use it for long or noisy recordings
- optimizations in the smart search mode, now implementing a rejection method based on amplitudes
- show the mean amplitude over time in the MATLAB GUI
- MATLAB is automatically closed when closing the MATLAB GUI
- mean rate is now displayed in the MATLAB GUI, for new datasets only
- spike times are now saved as uint32, for new datasets only
- various fixes in the docs
- improvements when peak detection is set on “both”
- message about cc_merge for low density probes
- message about smart search for long recordings
- various cosmetic changes
- add a conda app for anaconda navigator

Release 0.4.1

- fix a bug for converting millions of PCs to phy, getting rid of MPI limitation to int32
- fix bugs with install on Windows 10, forcing int64 while default is int32 even on 64bits platforms
- improved errors messages if wrong MCS headers are used
- Various cosmetic changes

Release 0.4

First realease of the software

Future plans and contributions

Here is a non-exhaustive list of the features that we are currently working on, and that should make it into future releases of the software

Real Time spike sorting

This is the most challenging task, and we are thinking about what is the best way to properly implement it. Such a real-time spike sorting for dense arrays is within reach, but several challenges need to be addressed to make it possible. Data will be read from memory streams, and templates will be updated on-the-fly. The plan is to have spatio-temporal templates tracking cells over time, at a cost of a small temporal lag that can not be avoided because of the template-matching step.

Better, faster, stronger

GPU kernels should be optimized to increase the speed of the algorithm, and we are always seeking for optimizations along the road. For Real-Time spike sorting, if we want it to be accurate for thousands of channels, any optimizations is welcome.

Contributions

If you have ideas, or if you want to contribute to the software, with the same idea that we should develop a proper and unified framework for semi-automated spike sorting, please do not hesitate to contact pierre.yger@inserm.fr . Currently, the code itself is not properly documented, as our main focus was to first get a stable working algorithm. Now that this goal is now achieved, we can dive more into software development and enhance its modularity.

CHAPTER 2

Launching the code

In this section, you will find all the information you need to be able to launch the code, and obtain results on any given dataset. To know more about how to visualize them, you will find everything in the following section

Quickstart

Running the algorithm

Copy your files

First, you will need to create a directory (we call it `path` – usually you put both the date of the experiment and the name of the person doing the sorting). Your data file should have a name like `path/mydata.extension`

Warning: Your data should not be filtered, and by default the filtering will be done only once **onto** the data. So you need to keep a copy elsewhere of you raw data. If you really do not want to filter data on site, you can use the `overwrite` parameter (see *documentation on the code* for more information).

Generate a parameter file

Before running the algorithm, you will always need to provide parameters, as a parameter file. Note that this parameter file has to be in the same folder than your data, and should be named `path/mydata.params`. If you have already yours, great, just copy it in the folder. Otherwise, just launch the algorithm, and the algorithm will ask you if you want to create a template one, that you have to edit before launching the code:

```
>> spyking-circus.py path/mydata.extension
##############################################################
#####          Welcome to the SpyKING CIRCUS             #####
#####                                                    #####
#####          Written by P.Yger and O.Marre             #####
```

```
##############################################################
The parameter file is not present!
You must have a file named path/mydata.params, properly configured,
in the same folder, with the data file.
Do you want SpyKING CIRCUS to create a template there? [y/n]
```

In the parameter file, you mostly have to change only informations in the `data` section (see *documentation on the code* for more information).

Run the algorithm

Then you should run the algorithm by typing the following command(s):

```
>> spyking-circus path/mydata.extension
```

It should take around the time of the recording to run – maybe a bit more. The typical output of the program will be something like:

```
##################################################################
#####              Welcome to the SpyKING CIRCUS             #####
#####                                                        #####
#####              Written by P.Yger and O.Marre             #####
##################################################################

Steps         : filtering, whitening, clustering, fitting
GPU detected  : False
Number of CPU : 1
Parallel HDF5 : True
Shared memory : True
Hostfile      : /home/pierre/spyking-circus/circus.hosts

##################################################################

-------------------------  Informations  -------------------------
| Number of recorded channels : 252
| Number of analyzed channels : 252
| Data type                   : uint16
| Sampling rate               : 10 kHz
| Header offset for the data  : 1794
| Duration of the recording   : 7 min 12 s
| Width of the templates      : 5 ms
| Spatial radius considered   : 250 um
| Waveform alignment          : True
| Skip strong artefacts       : False
| Template Extraction         : median-raw
------------------------------------------------------------------
-------------------------  Informations  -------------------------
| Filtering has already been done with cut off at 500Hz
------------------------------------------------------------------
Analyzing data to get whitening matrices and thresholds...
We found 20s without spikes for whitening matrices...
Because of whitening, we need to recompute the thresholds...
Searching spikes to construct the PCA basis...
100%|#####################################################
```

Note that you can of course change the number of CPU/GPU used, and also launch only a subset of the steps. See the help of the code to have more informations.

Using Several CPUs

To use several CPUs, you should have a proper installation of MPI, and a valid hostfile given to the program. See *documentation on MPI*. And then, you simply need to do, if *N* is the number of processors:

```
>> spyking-circus path/mydata.extension -c N
```

Using the GUI

Get the data

Once the algorithm has run on the data path/mydata.extension, you should have the following files in the directory path:

- `path/mydata/mydata.result.hdf5`
- `path/mydata/mydata.cluster.hdf5`
- `path/mydata/mydata.overlap.hdf5`
- `path/mydata/mydata.templates.hdf5`
- `path/mydata/mydata.basis.hdf5`

See the details here see *file formats* to know more how those files are structured.

Matlab GUI

To launch the MATLAB GUI provided with the software, you need of course to have a valid installation of MATLAB, and you should be able to simply do:

```
>> circus-gui-matlab path/mydata.extension
```

Python GUI

An experimental GUI derived from phy and made especially for template-matching based algorithms can be launched by doing:

```
>> spyking-circus path/mydata.extension -m converting
>> circus-gui-python path/mydata.extension
```

To enable it, you must have a valid installation of phy and phycontrib

To know more about the GUI section, see *documentation on the GUI*

Parameters

Display the helpers

To know what are all the parameters of the software, just do:

```
>> spyking-circus -h
```

To know what are all the file formats supported by the software, just do:

```
>> spyking-circus help -i
```

To know more what are the parameter of a given file format *X*, just do

```
>> spyking-circus X -i
```

Command line Parameters

The parameters to launch the program are:

- `-m` or `--method`

What are the steps of the algorithm you would like to perform. Defaults steps are:

1. filtering
2. whitening
3. clustering
4. fitting

Note that filtering is performed only once, and if the code is relaunched on the same data, a flag in the parameter file will prevent the code to filter twice. You can specify only a subset of steps by doing:

```
>> spyking-circus path/mydata.extension -m clustering,fitting
```

Note: Extra steps are available, such as `merging` (see the devoted section *documentation on Meta Merging*), or even more (*documentation on extra steps*).

- `-c` or `--cpu`

The number of CPU that will be used by the code, at least during the first three steps. Note that if CUDA is present, and if the GPU are not turned off (with -g 0), GPUs are always preferred to CPU during the fitting phase.

For example, just do:

```
>> spyking-circus path/mydata.extension -m clustering,fitting -c 10
```

- `-g` or `--gpu`

The number of GPU that will be used by the code during the fitting phase. If you have CUDA, but a slow GPU and a lot of CPUs (for example 10), you can disable the GPU usage by setting:

```
>> spyking-circus path/mydata.extension -g 0 -c 10
```

Warning: Currently, clusters with heterogeneous numbers of GPUs per nodes are not properly handled. Be in touch if interested by the functionality

- `-H` or `--hostfile`

The CPUs used depends on your MPI configuration. If you wan to configure them, you must provide a specific hostfile and do:

```
>> spyking-circus path/mydata.extension -c 10 -H nodes.hosts
```

To know more about the host file, see the MPI section *documentation on MPI*

- `-b` or `--batch`

The code can accept a text file with several commands that will be executed one after the other, in a batch mode. This is interesting for processing several datasets in a row. An example of such a text file `commands.txt` would simply be:

```
path/mydata1.extention -c 10
path/mydata2.extention -c 10 -m fitting
path/mydata3.extention -c 10 -m clustering,fitting,converting
```

Then simply launch the code by doing:

```
>> spyking-circus commands.txt -b
```

Warning: When processing files in a batch mode, be sure that the parameters file have been pre-generated. Otherwise, the code will hang asking you to generate them

- `-p` or `--preview`

To be sure that data are properly loaded before filtering everything on site, the code will load only the first second of the data, computes thresholds, and show you an interactive GUI to visualize everything. Please see the *documentation on Python GUI*

Note: The preview mode does not modify the data file!

- `-r` or `--result`

Launch an interactive GUI to show you, superimposed, the activity on your electrodes and the reconstruction provided by the software. This has to be used as a sanity check. Please see the *documentation on Python GUI*

- `-s` or `--second`

If the preview mode is activated, by default, it will show the first 2 seconds of the data. But you can specify an offset, in second, with this extra parameter such that the preview mode will display the signal in [second, second+2]

- `-o` or `--output`

If you want to generate synthetic benchmarks from a dataset that you have already sorted, this allows you, using the `benchmarking` mode, to produce a new file `output` based on what type of benchmarks you want to do (see `type`)

- `-t` or `--type`

While generating synthetic datasets, you have to chose from one of those three possibilities: `fitting`, `clustering`, `synchrony`. To know more about what those benchmarks are, see the *documentation on extra steps*

Note: Benchmarks will be better integrated soon into an automatic test suite, use them at your own risks for now. To know more about the additional extra steps, *documentation on extra steps*

Configuration File

The code, when launched for the first time, generates a parameter file. The default template used for the parameter files is the one located in `/home/user/spyking-circus/config.params`. You can edit it in advance if you are always using the same setup.

To know more about what is in the configuration file, *documentation on the configuration*

Designing your probe file

What is the probe file?

In order to launch the code, you must specify a mapping for your electrode, i.e you must tell the code how your recorded data can be mapped onto the physical space, and what is the spatial position of all your channels. Examples of such probe files (with the extension `.prb`) can be seen in the `probes` folder of the code. They will all look like the following one:

```
total_nb_channels = 32
radius            = 100

channel_groups = {
    1: {
        'channels': list(range(32)),
        'graph' : [],
        'geometry': {
            0 : [20, 0],
            1 : [20, 40],
            2 : [40, 210],
            3 : [40, 190],
            4 : [40, 150],
            5 : [40, 110],
            6 : [40, 70],
            7 : [40, 30],
            8 : [20, 160],
            9 : [20, 200],
            10 : [40, 50],
            11 : [40, 90],
            12 : [40, 130],
            13 : [40, 170],
            14 : [20, 120],
            15 : [20, 80],
            16 : [20, 100],
            17 : [20, 140],
            18 : [0, 170],
            19 : [0, 130],
            20 : [0, 90],
            21 : [0, 50],
            22 : [20, 220],
            23 : [20, 180],
            24 : [0, 30],
            25 : [0, 70],
            26 : [0, 110],
            27 : [0, 150],
            28 : [0, 190],
            29 : [0, 210],
            30 : [20, 60],
```

```
            31 : [20, 20],
        }
    }
}
```

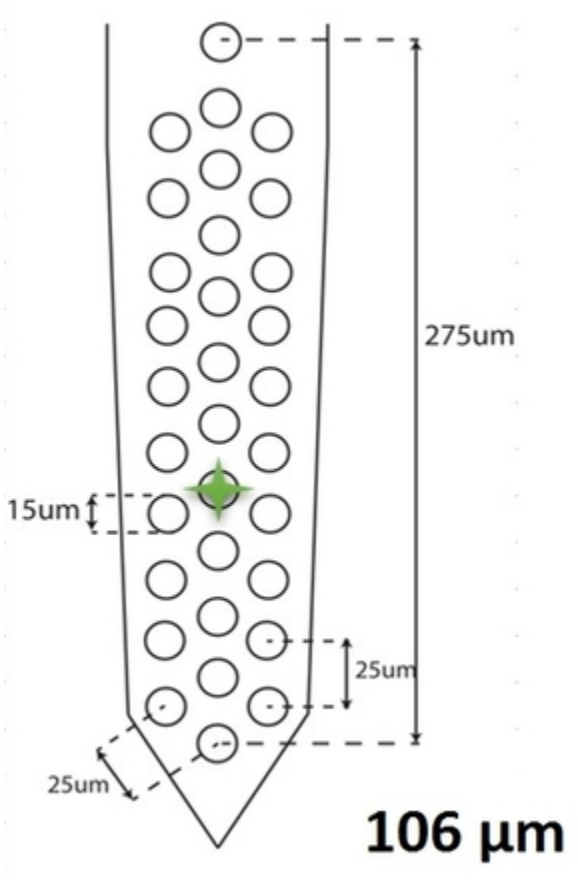


Fig. 2.1: An example of a probe mapping, taken from Adam Kampff

This `prb` format is inherited from the phy documentation, in order to ensure compatibility.

Key parameters

As you can see, an extra requirement of the SpyKING CIRCUS is that you specify, at the top of the probe file, two parameters:

- `total_nb_channels`: The total number of channels currently recorded. This has to be the number of rows in your data file
- `radius`: The default spatial extent [in um] of the templates that will be considered for that given probe. Note that for *in vitro* recording, such as the MEA with 252 channels, a spike can usually be seen in a physical radius of 250um. For *in vivo* data, 100um seems like a more reasonable value. You can change this value in the parameter file generated by the algorithm (see *documentation on the configuration file*)

Channel groups

The channel_group is a python dictionary where you'll specify, for every electrodes (you can have several of them), the exact geometry of all the recording sites on that probe, and what are the channels that should be processed by the algorithm. To be more explicit, in the previous example, there is one entry in the dictionary (with key 1), and this entry is itself a dictionary with three entries:

- `channels`: The list of the channels that will be considered by the algorithm. Note that even if your electrode has *N* channels, some can be discarded if they are not listed in this `channels` list.
- `graph`: Not used by the SpyKING CIRCUS, only here to ensure compatibility with phy
- `geometry`: This is where you have to specify all the physical positions of your channels. This is itself a dictionary, whose entries are the number of the channels, and whose values are the position [in um], of the recoding sites on your probe.

Note: You only need, in the `geometry` dictionary, to have entries for the channels you are listing in the `channels` list. The code only needs positions for analyzed channels

---

Examples

By default, during the install process, the code should copy some default probe files into `/home/user/spyking-circus/probes`. You can have a look at them.

How do deal with several shanks ?

There are two ways to simply handle several shanks:

- in the `.prb` file, you can create a single large channel group, where all the shanks are far enough (for example in the x direction), such that templates will not interact (based on the physical `radius`). If your radius is 200umm, for example, if you set x to 0 for the first shank, 300 for the second one, and so on, templates will be confined per shank.
- in the `.prb` file, you can also have several channel groups (see for example adrien.prb in the probes folder). What is done by the code, then, is that during internal computations templates are confined to each channel groups. However, for graphical purpose, when you'll use the GUI, the global x/y coordinates across all shanks are used. Therefore, if you do not want to have them plotted on top of each other, you still need to add a x/y padding for all of them.

Configuration File

This is the core of the algorithm, so this file has to be filled properly based on your data. Even if all key parameters of the algorithm are listed in the file, only few are likely to be modified by a non-advanced user. The configuration file is divided in several sections. For all those sections, we will review the parameters, and tell you what are the most important ones

Data

The data section is:

```
file_format     =                       # Can be raw_binary, openephys, hdf5, ... See >
↪> spyking-circus help -i for more info
stream_mode     = None                  # None by default. Can be multi-files, or␣
↪anything depending to the file format
mapping         = ~/probes/mea_252.prb  # Mapping of the electrode (see http://spyking-
↪circus.rtfd.ord)
suffix          =                       # Suffix to add to generated files
global_tmp      = True                  # should be False if local /tmp/ has enough␣
↪space (better for clusters)
overwrite       = True                  # If you want to filter or remove artefacts on␣
↪site. Data are duplicated otherwise
```

Warning: This is the most important section, that will allow the code to properly load your data. If not properly filled, then results will be wrong. Note that depending on your file_format, you may need to add here several parameters, such as `sampling_rate`, `data_dtype`, ... They will be requested if they can not be infered from the header of your data structure. To check if data are properly loaded, consider using *the preview mode* before launching the whole algorithm

Parameters that are most likely to be changed:

- `file_format` You must select a supported file format (see *What are the supported formats*) or write your own wrapper (see *Write your own data format*)
- `mapping` This is the path to your probe mapping (see *How to design a probe file*)
- `global_temp` If you are using a cluster with NFS, this should be False (local /tmp/ will be used by every nodes)
- `stream_mode` If streams in you data (could be multi-files, or even in the same file) should be processed together (see *Using multi files*)
- `overwrite` If True, data are overwritten during filtering, assuming the file format has write access. Otherwise, an external raw_binary file will be created during the filtering step, if any.

Detection

The detection section is:

```
radius          = auto      # Radius [in um] (if auto, read from the prb file)
N_t             = 5         # Width of the templates [in ms]
spike_thresh    = 6         # Threshold for spike detection
peaks           = negative  # Can be negative (default), positive or both
matched-filter  = False     # If True, we perform spike detection with matched filters
matched_thresh  = 5         # Threshold for detection if matched filter is True
alignment       = True      # Realign the waveforms by oversampling
```

Parameters that are most likely to be changed:

- `N_t` The temporal width of the templates. For *in vitro* data, 5ms seems a good value. For *in vivo* data, you should rather use 3 or even 2ms
- `radius` The spatial width of the templates. By default, this value is read from the probe file. However, if you want to specify a larger or a smaller value [in um], you can do it here
- `spike_thresh` The threshold for spike detection. 6-7 are good values
- `peaks` By default, the code detects only negative peaks, but you can search for positive peaks, or both

- `matched-filter` If activated, the code will detect smaller spikes by using matched filtering
- `matched_threhs` During matched filtering, the detection threshold
- `alignment` By default, during clustering, the waveforms are realigned by oversampling at 5 times the sampling rate and using bicubic spline interpolation

Filtering

The filtering section is:

```
cut_off        = 500, auto # Min and Max (auto=nyquist) cut off frequencies for the␣
↪band pass butterworth filter [Hz]
filter         = True      # If True, then a low-pass filtering is performed
remove_median  = False     # If True, median over all channels is substracted to each␣
↪channels (movement artefacts)
```

Warning: The code performs the filtering of your data writing on the file itself. Therefore, you `must` have a copy of your raw data elsewhere. Note that as long as your keeping the parameter files, you can relaunch the code safely: the program will not filter multiple times the data, because of the flag `filter_done` at the end of the configuration file.

Parameters that are most likely to be changed:

- `cut_off` The default value of 500Hz has been used in various recordings, but you can change it if needed. You can also specify the upper bound of the Butterworth filter
- `filter` If your data are already filtered by a third program, turn that flag to False
- `remove_median` If you have some movement artefacts in your *in vivo* recording, and want to substract the median activity over all analysed channels from each channel individually

Triggers

The triggers section is:

```
trig_file      =           # If external stimuli need to be considered as putative␣
↪artefacts (see documentation)
trig_windows   =           # The time windows of those external stimuli [in ms]
clean_artefact = False     # If True, external artefacts induced by triggers will be␣
↪suppressed from data
trig_unit      = ms        # The unit in which times are expressed: can be either ms␣
↪or timestep
make_plots     =           # Generate sanity plots of the averaged artefacts [Nothing␣
↪or None if no plots]
```

Parameters that are most likely to be changed:

- `trig_file` The path to the file where your artefact times and labels. See *how to deal with stimulation artefacts*
- `trig_windows` The path to file where your artefact temporal windows. See *how to deal with stimulation artefacts*
- `clean_artefact` If you want to remove any stimulation artefacts, defined in the previous file. See *how to deal with stimulation artefacts*

- `make_plots` The default format to save the plots of the artefacts, one per artefact, showing all channels. You can set it to None if you do not want any
- `trig_unit` If you want times/duration in the `trig_file` and `trig_windows` to be in timestep or ms

Whitening

The whitening section is:

```
chunk_size      = 60        # Size of the data chunks [in s]
safety_time     = 1         # Temporal zone around which templates are isolated [in ms]
temporal        = False     # Perform temporal whitening
spatial         = True      # Perform spatial whitening
max_elts        = 10000     # Max number of events per electrode (should be compatible␣
↪with nb_elts)
nb_elts         = 0.8       # Fraction of max_elts that should be obtained per␣
↪electrode [0-1]
output_dim      = 5         # Can be in percent of variance explain, or num of␣
↪dimensions for PCA on waveforms
```

Parameters that are most likely to be changed:

- `output_dim` If you want to save some memory usage, you can reduce the number of features kept to describe a waveform.
- `chunk_size` If you have a very large number of electrode, and not enough memory, you can reduce it

Clustering

The clustering section is:

```
extraction      = median-raw # Can be either median-raw (default), median-pca, mean-
↪pca, mean-raw, or quadratic
safety_space    = True       # If True, we exclude spikes in the vicinity of a␣
↪selected spikes
safety_time     = 1          # Temporal zone around which templates are isolated [in␣
↪ms]
max_elts        = 10000      # Max number of events per electrode (should be␣
↪compatible with nb_elts)
nb_elts         = 0.8        # Fraction of max_elts that should be obtained per␣
↪electrode [0-1]
nclus_min       = 0.002      # Min number of elements in a cluster (given in␣
↪percentage)
max_clusters    = 10         # Maximal number of clusters for every electrodes
nb_repeats      = 3          # Number of passes used for the clustering
make_plots      =            # Generate sanity plots of the clustering
sim_same_elec   = 3          # Distance within clusters under which they are re-merged
cc_merge        = 0.975      # If CC between two templates is higher, they are merged
dispersion      = (5, 5)     # Min and Max dispersion allowed for amplitudes [in MAD]
smart_search    = True       # Parameter to activate the smart search mode
smart_select    = False      # Experimental: activate the smart selection of centroids␣
↪(max_clusters is ignored)
noise_thr       = 0.8        # Minimal amplitudes are such than amp*min(templates) <␣
↪noise_thr*threshold
remove_mixture = True        # At the end of the clustering, we remove mixtures of␣
↪templates
```

Note: This is the a key section, as bad clustering will implies bad results. However, the code is very robust to parameters changes.

Parameters that are most likely to be changed:

- `extraction` The method to estimate the templates. `Raw` methods are slower, but more accurate, as data are read from the files. `PCA` methods are faster, but less accurate, and may lead to some distorted templates. `Quadratic` is slower, and should not be used.
- `max_elts` The number of elements that every electrode will try to collect, in order to perform the clustering
- `nclus_min` If you have too many clusters with few elements, you can increase this value. This is expressed in percentage of collected spike per electrode. So one electrode collecting *max_elts* spikes will keep clusters with more than *nclus_min.max_elts*. Otherwise, they are discarded
- `max_clusters` This is the maximal number of cluster that you expect to see on a given electrode. For *in vitro* data, 10 seems to be a reasonable value. For *in vivo* data and dense probes, you should set it to 10-15. Increase it only if the code tells you so.
- `nb_repeats` The number of passes performed by the algorithm to refine the density landscape
- `smart_search` By default, the code will collect only a subset of spikes, randomly, on all electrodes. However, for long recordings, or if you have low thresholds, you may want to select them in a smarter manner, in order to avoid missing the large ones, under represented. If the smart search is activated, the code will first sample the distribution of amplitudes, on all channels, and then implement a rejection algorithm such that it will try to select spikes in order to make the distribution of amplitudes more uniform. This can be very efficient, and may become True by default in future releases.
- `smart_select` This option (experimental) should boost the quality of the clustering, by selecting the centroids in a automatic manner. If activated the `max_clusters` parameter is ignored
- `cc_merge` After local merging per electrode, this step will make sure that you do not have duplicates in your templates, that may have been spread on several electrodes. All templates with a correlation coefficient higher than that parameter are merged. Remember that the more you merge, the faster is the fit
- `dispersion` The spread of the amplitudes allowed, for every templates, around the centroid.
- `remove_mixture` By default, any template that can be explained as sum of two others is deleted.
- `make_plots` By default, the code generates sanity plots of the clustering, one per electrode.

Fitting

The fitting section is:

```
chunk          = 1          # Size of chunks used during fitting [in second]
gpu_only       = True       # Use GPU for computation of b's AND fitting
amp_limits     = (0.3, 30)  # Amplitudes for the templates during spike detection
amp_auto       = True       # True if amplitudes are adjusted automatically for every␣
↪templates
max_chunk      = inf        # Fit only up to max_chunk
collect_all    = False       # If True, one garbage template per electrode is created,␣
↪to store unfitted spikes
```

Parameters that are most likely to be changed:

- `chunk` again, to reduce memory usage, you can reduce the size of the temporal chunks during fitting. Note that it has to be one order of magnitude higher than the template width `N_t`
- `gpu_only` By default, all operations will take place on the GPU. However, if not enough memory is available on the GPU, then you can turn this flag to False.
- `max_chunk` If you just want to fit the first *N* chunks, otherwise, the whole file is processed
- `collect_all` If you want to also collect all the spike times at which no templates were fitted. This is particularly useful to debug the algorithm, and understand if something is wrong on a given channel

Merging

The merging section is:

```
cc_overlap      = 0.5       # Only templates with CC higher than cc_overlap may be␣
↪merged
cc_bin          = 2         # Bin size for computing CC [in ms]
correct_lag     = False     # If spikes are aligned when merging. May be better for␣
↪phy usage
```

To know more about how those merges are performed and how to use this option, see *Automatic Merging*. Parameters that are n

- `correct_lag` By default, in the meta-merging GUI, when two templates are merged, the spike times of the one removed are simply added to the one kept, without modification. However, it is more accurate to shift those spike, in times, by the temporal shift that may exist between those two templates. This will lead to a better visualization in phy, with more aligned spikes

Converting

The converting section is:

```
erase_all       = True      # If False, a prompt will ask you to export if export has␣
↪already been done
sparse_export   = False     # If True, data for phy are exported in a sparse format.␣
↪Need recent version of phy
export_pcs      = prompt    # Can be prompt [default] or in none, all, some
export_all      = False     # If True, unfitted spikes will be exported as the last Ne␣
↪templates
```

Parameters that are most likely to be changed:

- `erase_all` If you want to always erase former export, and skip the prompt
- `sparse_export` If you have a large number of templates or a very high density probe, you should use the sparse format for phy
- `export_pcs` If you already know that you want to have all, some, or no PC and skip the prompt
- `export_all` If you used the `collect_all` mode in the `[fitting]` section, you can export unfitted spike times to phy. In this case, the last *N* templates, if *N* is the number of electrodes, are the garbage collectors.

Extracting

The extracting section is:

```
safety_time     = 1          # Temporal zone around which spikes are isolated [in ms]
max_elts        = 10000      # Max number of events per templates (should be compatible␣
↪with nb_elts)
nb_elts         = 0.8        # Fraction of max_elts that should be obtained per␣
↪electrode [0-1]
output_dim      = 5          # Percentage of variance explained while performing PCA
cc_merge        = 0.975      # If CC between two templates is higher, they are merged
noise_thr       = 0.8        # Minimal amplitudes are such than amp*min(templates) <␣
↪noise_thr*threshold
```

This is an experimental section, not used by default in the algorithm, so nothing to be changed here

Validating

The validating section is:

```
nearest_elec    = auto       # Validation channel (e.g. electrode closest to the ground␣
↪truth cell)
max_iter        = 200        # Maximum number of iterations of the stochastic gradient␣
↪descent (SGD)
learning_rate   = 1.0e-3     # Initial learning rate which controls the step-size of␣
↪the SGD
roc_sampling    = 10         # Number of points to estimate the ROC curve of the BEER␣
↪estimate
make_plots      = png        # Generate sanity plots of the validation [Nothing or None␣
↪if no plots]
test_size       = 0.3        # Portion of the dataset to include in the test split
radius_factor   = 0.5        # Radius factor to modulate physical radius during␣
↪validation
juxta_dtype     = uint16     # Type of the juxtacellular data
juxta_thresh    = 6          # Threshold for juxtacellular detection
juxta_valley    = False      # True if juxta-cellular spikes are negative peaks
```

Please get in touch with us if you want to use this section, only for validation purposes. This is an implementation of the *BEER metric*

Supported File Formats

To get the list of supported file format, you need to do:

```
>> spyking-circus help -i
-------------------------  Informations  -------------------------
| The file formats that are supported are:
|
| -- PLEXON (read only)
|       Extensions       : .plx
|       Supported streams: multi-files
| -- KWD (read/write)
|       Extensions       : .kwd
|       Supported streams: multi-files, single-file
| -- BRW (read/write)
|       Extensions       : .brw
|       Supported streams: multi-files
| -- MCD (read only)
```

```
|       Extensions       : .mcd
|       Supported streams: multi-files
| -- MCS_RAW_BINARY (read/parallel write)
|       Extensions       : .raw, .dat
|       Supported streams: multi-files
| -- NWB (read/write)
|       Extensions       : .nwb, .h5, .hdf5
|       Supported streams: multi-files
| -- RHD (read/write)
|       Extensions       : .rhd
|       Supported streams: multi-files
| -- OPENEPHYS (read/parallel write)
|       Extensions       : .openephys
|       Supported streams: multi-files
| -- HDF5 (read/write)
|       Extensions       : .h5, .hdf5
|       Supported streams: multi-files
| -- NIX (read/write)
|       Extensions       : .nix, .h5, .hdf5
|       Supported streams: multi-files
| -- BLACKROCK (read only)
|       Extensions       : .nev
|       Supported streams: multi-files
| -- RAW_BINARY (read/parallel write)
|       Extensions       :
|       Supported streams: multi-files
| -- NUMPY (read/parallel write)
|       Extensions       : .npy
|       Supported streams: multi-files
| -- ARF (read/write)
|       Extensions       : .arf, .hdf5, .h5
|       Supported streams: multi-files, single-file
-----------------------------------------------------------------
```

This list will tell you what are the wrappers available, and you need to specify one in your configuration file with the `file_format` parameter in the `[data]` section. To know more about the mandatory/optional parameters for a given file format, you should do:

```
>> spyking-circus raw_binary -i
-------------------------  Informations  -------------------------
| The parameters for RAW_BINARY file format are:
|
| -- sampling_rate -- <type 'float'> [** mandatory **]
| -- data_dtype -- <type 'str'> [** mandatory **]
| -- nb_channels -- <type 'int'> [** mandatory **]
|
| -- data_offset -- <type 'int'> [default is 0]
| -- dtype_offset -- <type 'str'> [default is auto]
| -- gain -- <type 'int'> [default is 1]
-----------------------------------------------------------------
```

Note: Depending on the file format, the parameters needed in the `[data]` section of the parameter file can vary. Some file format are self-contained, while some others need extra parameters to reconstruct the data. For all the needed parameters, you need to add in the `[data]` section of the parameter file a line with `parameter = value`

Warning: As said after, only file format derived from `raw_binary`, and without streams are currently supported by the phy and MATLAB GUI, if you want to see the raw data. All other views, that do not depend on the raw data, will stay the same, so you can still sort your data.

Neuroshare support

Some of the file formats (plexon, blackrock, multi-channel systems, ..) can be accessed only if you have the neuroshare library installed. Note that despite a great simplicity of use, this library provides only slow read access and no write access to the file formats. Therefore, this is not an efficient wrapper, and it may slow down considerably the code. Feel free to contribute if you have better ideas about what to do!

HDF5-like file

This should be easy to implement any HDF5-like file format. Some are already available, feel free to add yours. Note that to allow parallel write with HDF5, you must have a version of HDF5 compiled with the MPI option activated. This means that you need to do a manual install.

Raw binary File

The simplest file format is the raw_binary one. Suppose you have *N* channels

$$c_0, c_1, ..., c_N$$

And if you assume that $c_i(t)$ is the value of channel c_i at time *t*, then your datafile should be a raw file with values

$$c_0(0), c_1(0), ..., c_N(0), c_0(1), ..., c_N(1), ...c_N(T)$$

This is simply the flatten version of your recordings matrix, with size *N* x *T*

Note: The values can be saved in your own format (`int16`, `uint16`, `int8`, `float32`). You simply need to specify that to the code

As you can see by typing:

```
>> spyking-circus raw_binary -i
-------------------------  Informations  -------------------------
| The parameters for RAW_BINARY file format are:
|
| -- sampling_rate -- <type 'float'> [** mandatory **]
| -- data_dtype -- <type 'str'> [** mandatory **]
| -- nb_channels -- <type 'int'> [** mandatory **]
|
| -- data_offset -- <type 'int'> [default is 0]
| -- dtype_offset -- <type 'str'> [default is auto]
| -- gain -- <type 'int'> [default is 1]
-----------------------------------------------------------------
```

There are some extra and required parameters for the raw_binary file format. For example, you must specify the sampling rate `sampling_rate`, the data_dtype (`int16`, `float32`, ...) and also the number of channels `nb_channels`. The remaining parameters are optional, i.e. if not provided, default values written there will be used. So the `mydata.params` file for a `mydata.dat` raw binary file will have the following params in the `[data]` section:

```
file_format    = raw_binary
sampling_rate  = XXXX
data_dtype     = XXXX # should be int16,uint16,float32,...
nb_channels    = XXXX # as it can not be guessed from the file, it has to be specified
data_offset    = XXXX # Optional, if a header with a fixed size is present
gain           = XXXX # Optional, if you want a non unitary gain for the channels
```

Warning: The `raw_binary` file format is the default one used internally by SpyKING CIRCUS when the flag `overwrite` is set to `False`. This means several things

- data are saved as `float32`, so storage can be large
- we can not handle properly t_start parameters if there are streams in the original data. Times will be continuous
- this is currently the **only** file format properly supported by phy and MATLAB GUIs, if you want to see the raw data

Sanity plots

In order to have a better feedback on what the algorithm is doing, and especially the clustering phase, the code can produce sanity plots that may be helpful to troubleshoot. This is the flag `make_plots` in the `clustering` section of the parameter files (see the configuration section *documentation on MPI*). All plots will be stored in the folder `path/mydata/plots`

Note: If you do not care about those plots, you can set to `None` the `make_plots` entries in the configuration file, and this will speed up the algorithm

View of the activity

The best way to visualize the activity on your electrodes, and to see if data are properly loaded or if results are making any sense is to use the devoted python GUI and the preview mode (see the visualization section *on Python GUI*)

Views of the Clusters

During the clustering phase, the algorithm will save files names `cluster_i` where *i* is the number of the electrode. A typical plot will look like that

On the two plots in the left column, you can see the rho vs delta plots (see [Rodriguez et Laio, 2014]). Top plots shows the centroids that have been selected, and bottom plots shows in red all the putative centers that were considered by the algorithm.

On the 4 plots on the rights, this is a 3D projection of all the spikes collected by that electrode, projected along different axes: x vs y, y vs z and x vs z.

Note: If, in those plots, you see clusters that you would have rather split, and that do not have different color, then this is likely that the clustering algorithm had wrong parameters. Remember that in the configuration file `max_clusters`

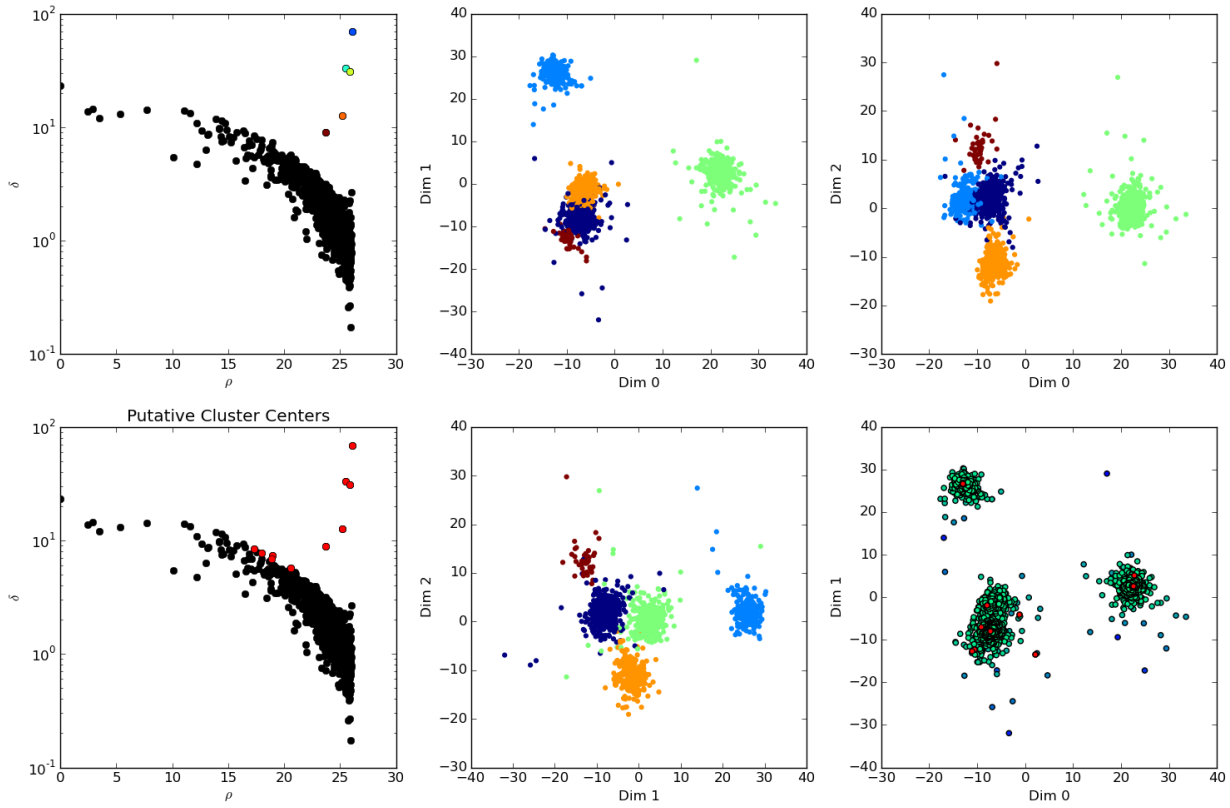


Fig. 2.2: A view on the clusters detected by the algorithm, on a given electrode

controls the maximal number of clusters per electrodes that will be searched (so you may want to increase it if clustering is not accurate enough), and that `sim_same_elec` will control how much similar clusters will be merged. So again, decrease it if you think you are losing some obvious clusters.

Views of the waveforms

At the end of the clustering phase, the algorithm will save files names `waveform_i` where *i* is the number of the electrode. A typical plot will look like that

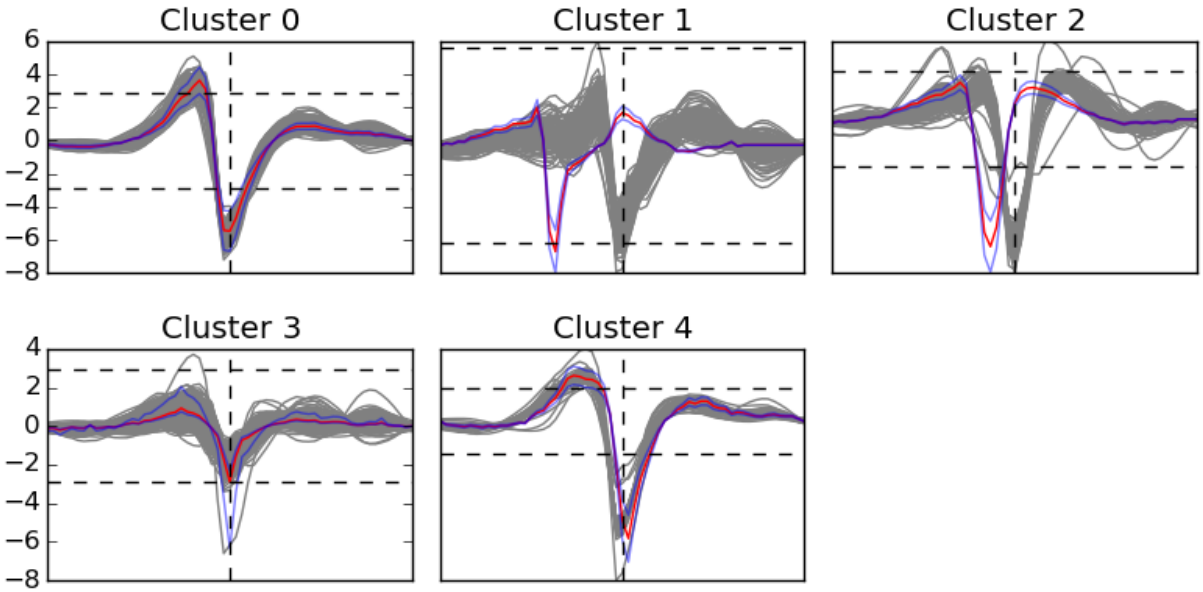


Fig. 2.3: A view on the templates, on a given electrode

On this plot, you should get an insight on the templates that have been computed out of the clustering phase. For all the clusters detected on that given electrode, you should see all the waveforms peaking on that particular electrode, and the template, in red (in blue, this is the min and max amplitudes allowed during the fitting procedure). Note that if template is not aligned with the waveforms, this is normal. The templates are aligned on the electrodes were they have an absolute min. Here you are just looking at them on a particular electrode. The key point is that, as you can see, templates should all go below threshold on that particular electrode (dash-dotted line).

Processing streams of data

It is often the case that, during the same recording session, the experimentalist records only some temporal chunks and not the whole experiment. However, because the neurons are the same all over the recording, it is better to process them as a single datafile. The code can handle such streams of data, either from multiple sources (several data files), or within the same source if supported by the file format (chunks in a single file).

Chunks spread over several files

You can use the `multi-files` stream mode in the `[data]` section.

Note: If you just want to process several *independent* files, coming from different recording sessions, you need to use the batch mode (see *the documentation on the parameters*)

For the sake of clarity, we assume that all your files are labelled

- `mydata_0.extension`
- `mydata_1.extension`
- ...
- `mydata_N.extension`

Launch the code on the first file:

```
>> spyking-circus mydata_0.extension
```

The code will create a parameter file, `mydata_0.params`. Edit the file, and in the `[data]` section, set `stream_mode` to `multi-files`. Relaunch the code on the first file only:

```
>> spyking-circus mydata_0.extension
```

The code will now display something like:

```
##################################################################
#####              Welcome to the SpyKING CIRCUS             #####
#####                                                        #####
#####              Written by P.Yger and O.Marre             #####
##################################################################

Steps         : fitting
GPU detected  : True
Number of CPU : 12
Number of GPU : 6
Shared Memory : True
Parallel HDF5 : False
Hostfile      : /home/spiky/spyking-circus/circus.hosts

##################################################################

-------------------------  Informations  -------------------------
| Number of recorded channels : 256
| Number of analyzed channels : 256
| Data type                   : uint16
| Sampling rate               : 20 kHz
```

```
| Header offset for the data  : 1881
| Duration of the recording   : 184 min 37 s
| Width of the templates      : 5 ms
| Spatial radius considered   : 250 um
| Stationarity                : True
| Waveform alignment          : True
| Skip strong artefacts       : True
| Template Extraction         : median-raw
| Streams                     : multi-files (19 found)
---------------------------------------------------------------
```

The key line here is the one stating that the code has detected 19 files, and will process them as a single one.

Note: The multi-files mode assumes that all files have the same properties: mapping, data type, data offset, ... It has to be the case if they are all coming from the same recording session

---

While running, in its first phase (filtering), two options are possible:

- if your file format allows write access, and `overwrite` is set to `True` in the `data` section, then every individual data file will be overwritten and filtered on site
- if your file format does not allow write access, or `overwrite` is `False`, the code will filter and concatenate all files into a new file, saved as a `float32` binary file called `mydata_all_sc.extension`. Templates are then detected onto this single files, and fitting is also applied onto it.

Chunks contained in the same datafile

For more complex data structures, several recordings sessions can be saved within the same datafile. Assuming the file format allows it (see *the documentation on the file formats*), the code can still stream all those chunks of data in order to process them as a whole. To do so, use exactly the same procedure as below, except that the `stream_mode` may be different, for example `single-file`.

Visualizing results from several streams

Multi-files

As said, results are obtained on a single file `mydata_all.extension`, resulting of the concatenation of all the individual files. So when you are launching the GUI:

```
>> circus-gui-matlab mydata_0.extension
```

what you are seeing are *all* the spikes on *all* files. Here you can delete/merge templates, see the devoted GUI section for that (*GUI*). Note that you need to process data in such a manner, because otherwise, if looking at all results individually, you would have a very hard time keeping track of the templates over several files. Plus, you would not get all the information contained in the whole recording (the same neuron could be silent during some temporal chunks, but spiking during others).

Getting individual results from streams

Once your manual sorting session is done, you can simply split the results in order to get one result file per data file. To do so, simply launch:

```
>> circus-multi mydata_0.extension
```

This will create several files

- `mydata_0.results.hdf5`
- `mydata_1.results.hdf5`
- ...
- `mydata_N.results.hdf5`

In each of them, you'll find the spike times of the given streams, between *0* and *T*, if *T* is the length of file *i*.

Dealing with stimulation artefacts

Sometimes, because of external stimulation, you may end up having some artefacts on top of your recordings. For example, in case of optogenetic stimulation, shinning light next to your recording electrode is likely to contaminate the recording. The code has a built-in mechanism to deal with those artefacts, in the `triggers` section of the parameter file. In a nutshell, the code will, from a list of stimulation times, compute the average artefact, and substract it automatically to the signal during the filtering procedure.

Specifying the stimulation times

In a first text file, you must specify all the times of your artefacts, identified by a given identifier. The times can be given in ms or in timesteps, and this can be changed with the `trig_unit` parameter. By default, they are assumed to be in ms. For example, imagine you have 2 different stimulation protocols, each one inducing a different artefact. The text file will look like:

```
0 500.2
1 1000.2
0 1500.3
1 2000.1
...
0 27364.1
1 80402.4
```

This means that stim 0 is displayed at 500.2ms, then stim 1 at 1000.2ms, and so on. All times in the text file are in ms, and you must use one line per time. Use `trig_unit` if you want to give times in timesteps.

Specifying the time windows

In a second text file, you must tell the algorithm what is the time window you want to consider for a given artefact. Using the same example, and assuming that stim 0 produces an artefact of 100ms, while stim 1 produces a longer artefact of 510ms, the file should look like:

```
0 100
1 510
```

Here, again, use `trig_unit` if you want to provide times in timesteps.

How to use it

Once those two files have been created, you should provide them in the `[triggers]` section of the code (see *here*). Note that by default, the code will produce one plot by artefact, showing its temporal time course on all channels, during the imposed time window. This is what is substracted, at all the given times for this unique stimulation artefact.

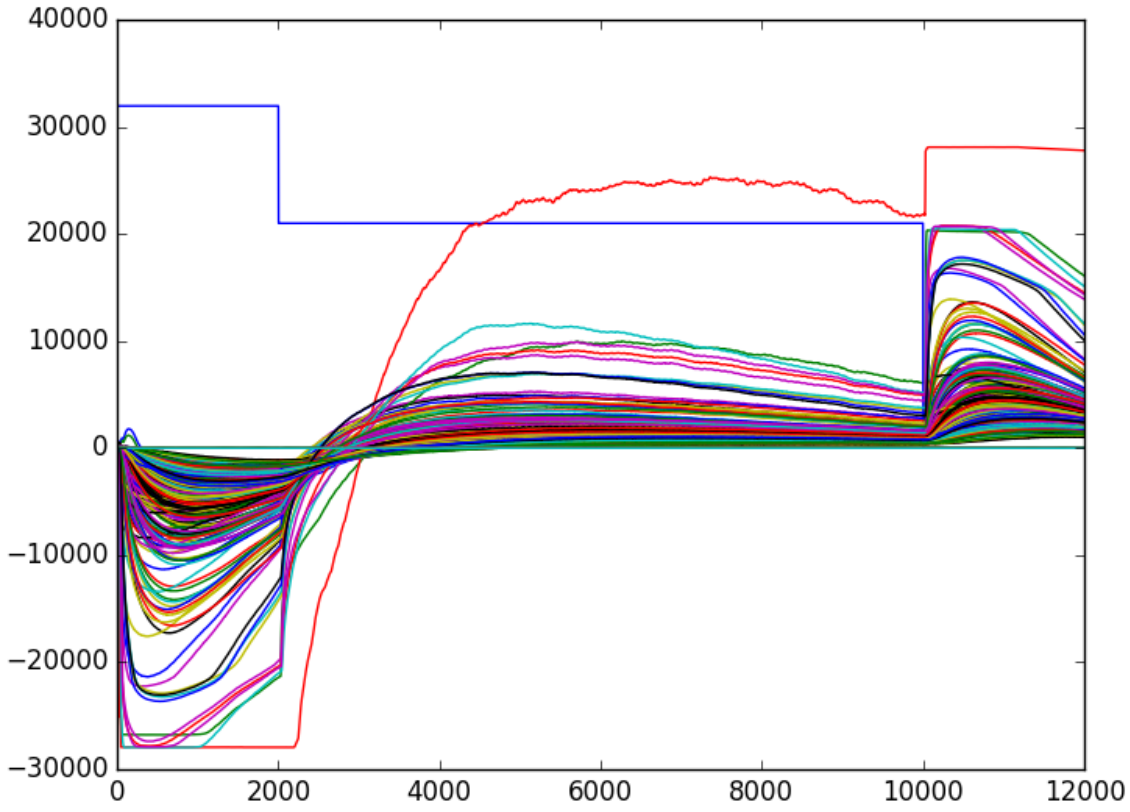


Fig. 2.4: Example of a stimulation artefact on a 252 MEA, substracted during the filtering part of the algorithm.

Note: If, for some reasons, you want to relaunch this step (too small time windows, not enough artefacts, ...) you will need to copy again the raw data before relaunching the filtering. This is because remember that the raw data are *always* filtered on-site.

Automatic Merging

Need for an meta merging step

Because for high number of channels, the chance that a cell can be split among several templates are high, one need to merge putative templates belonging to the same cells. This is a classical step in most of the spike sorting technique,

and traditionally, this step was performed by a human operator, reviewing all templates one by one. Problem is that with the new generation of dense probes that the code can handle (4225 channels), the output of the algorithm can lead to more than 1000 templates, and one can not expect a human to go through all pairs iteratively.

To automatize the procedure, we developed a so-called meta-merging step that will allow to quickly identify pairs of templates that have to be merged. To do so, first, we consider only pairs that have a similarity between their templates higher than `cc_overlap`. This allow not to considerate all the possible pairs, but only those that are likely to be the same cells, because their templates are similar.

Comparison of CrossCorrelograms

Then, for all those pairs of cells, we are computing the cross-correlation function in a time window of [-100, 100] ms, with a particular time bin `cc_bin`. The rationale behind is that a pair of template that should be merged should have a dip in the center of its cross-correlogram. To quantify that in an automated manner, we compute a control cross-correlogram in the same window of interest, but by reverting in time the spikes of cell 2. This allow us to compare the normal cross-correlogram between the two cells to a "control" one, keeping the same amount of correlation (see Figure).

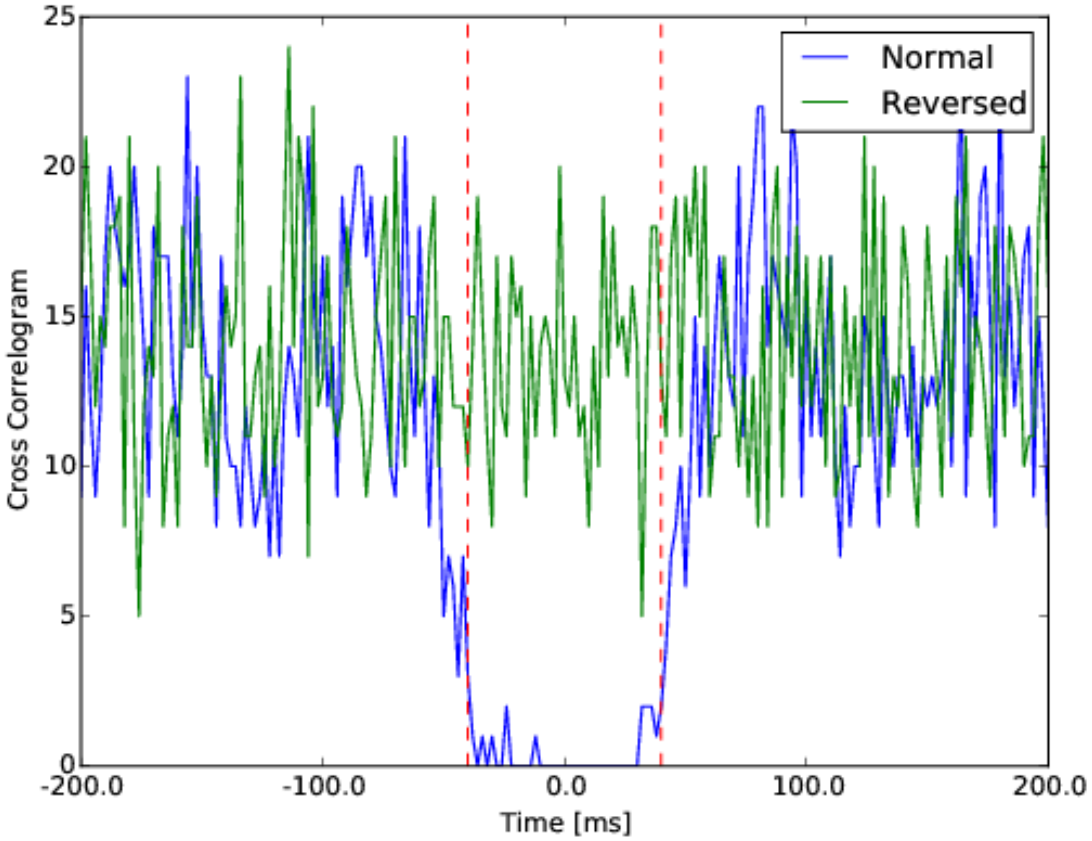


Fig. 2.5: Difference between a normal cross-correlogram for a given pair of cells, and a control version where the spikes from the second cells are reversed in time. The center area in between the red dash dotted line is the one of interest.

To quantify the dip, we measure the difference between the cross correlogram and its shuffled version in a window of

interest [`-cc_average`, `cc_average`].

An iterative procedure with a dedicated GUI

We design a Python GUI to quickly visualize all those values and allow human to quickly performs all merges that need to be done. To launch it, with *N* processors, you need to do:

```
>> spykig-circus mydata.extension -m merging -c N
```

The GUI is still an ongoing work, so any feedbacks are welcome, but the idea is to show, in a single plot, all the putative pairs of cells that have to be merged. As can be seen in the top left panel, every point is a pair of neuron, and x-axis in the upper left panel shows the template similarity (between `cc_merge` and 1), while y-axis show the normalized difference between the control CC and the normal CC (see above). In the bottom left plot, this is the same measure on the y-axis, while the x-axis only shows the CC of the Reverse Cross-Correlogram. **Any pairs along the diagonal are likely to be merged**

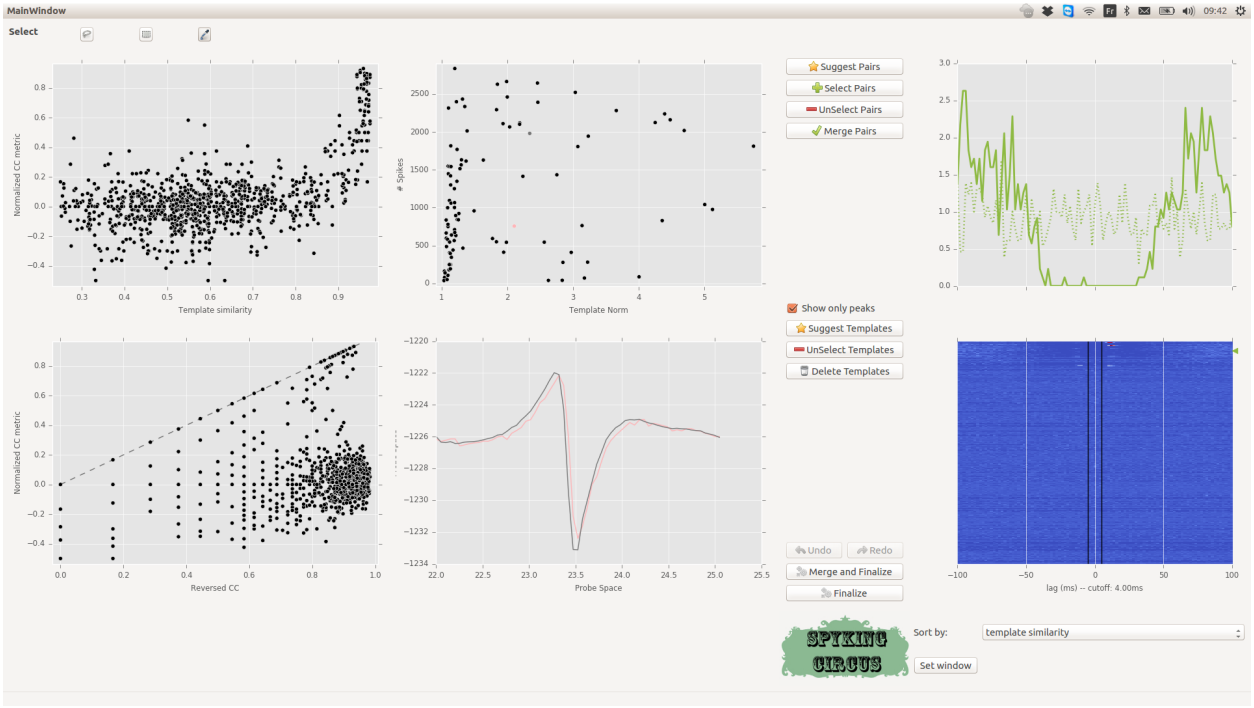


Fig. 2.6: Meta-merging GUI

Selecting pairs

Each time you click on a given pairs (or select a group of them with the rectangle or lasso selector), the corresponding Cross-Correlogram are shown in the top right panel (and in dash-dotted line, this is the control). As you can see, there is a clear group of pairs that have a high template similarity > 0.9, and a high value for the CC metric >0.5. So we can select some of them

If you think that all those pairs should be merged, you just need to click on the `Select` Button, and then on `Merge`. Once the merge is done, the GUI will recompute values and you can iterate the process

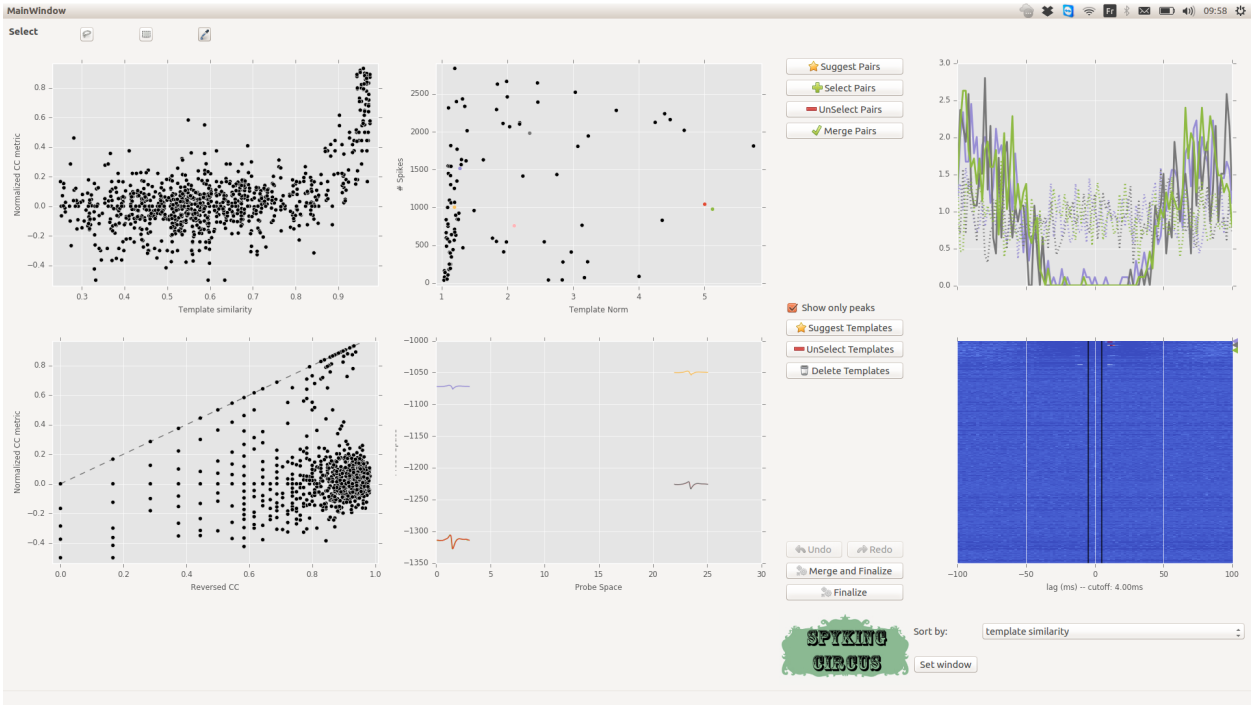


Fig. 2.7: Meta-merging GUI with several pairs that are selected

Note: The `Suggest Pairs` button suggests you pairs of neurons that have a template similarity higher than 0.9, and a high value for the CC metric

Changing the lag

By default, the CC metric is computed within a temporal window of [-5, 5] ms. But this value can be changed if you click on the `Set Window` Button. In the bottom right panel, you see all the CC for all pairs. You can change the way you want them to be sorted, and you can also click there to select some particular pairs.

Correcting for temporal lags while merging templates

By default, in the GUI, when a merge between two templates is performed, the spikes of the destroyed template are just assigned to the one that is kept. This is a valid assumption is most cases. However, if you want to be more accurate, you need to take into account a possible time shift between the two templates. This is especially True if you are detecting both positive and negative peaks. If a template is large enough to cross both positive and negative thresholds, two time shifted versions of the same template could exist. One will be centered on the positive peak, and one centered on the negative peak. So when you merge them, you need to apply to the spikes this time shift between the templates. This can be done if you set the `correct_lag` flag in the `[merging]` section of the parameter file to `True`.

Exploring Templates

In the middle, top plot, you can see on the x-axis the ratio between the peak of the template, and the detection threshold on its preferred electrode, and on the y-axis the number of spikes for that given templates. If you click on those points,

you'll see in the middle bottom plot the template waveform on its preferred electrode.

Note: The `Suggest Templates` button suggests you templates that have a peak below or just at the detection threshold. Those templates can exist, because of noise during the clustering phase. They are likely to be False templates, because the detection thresholds may have been set too low

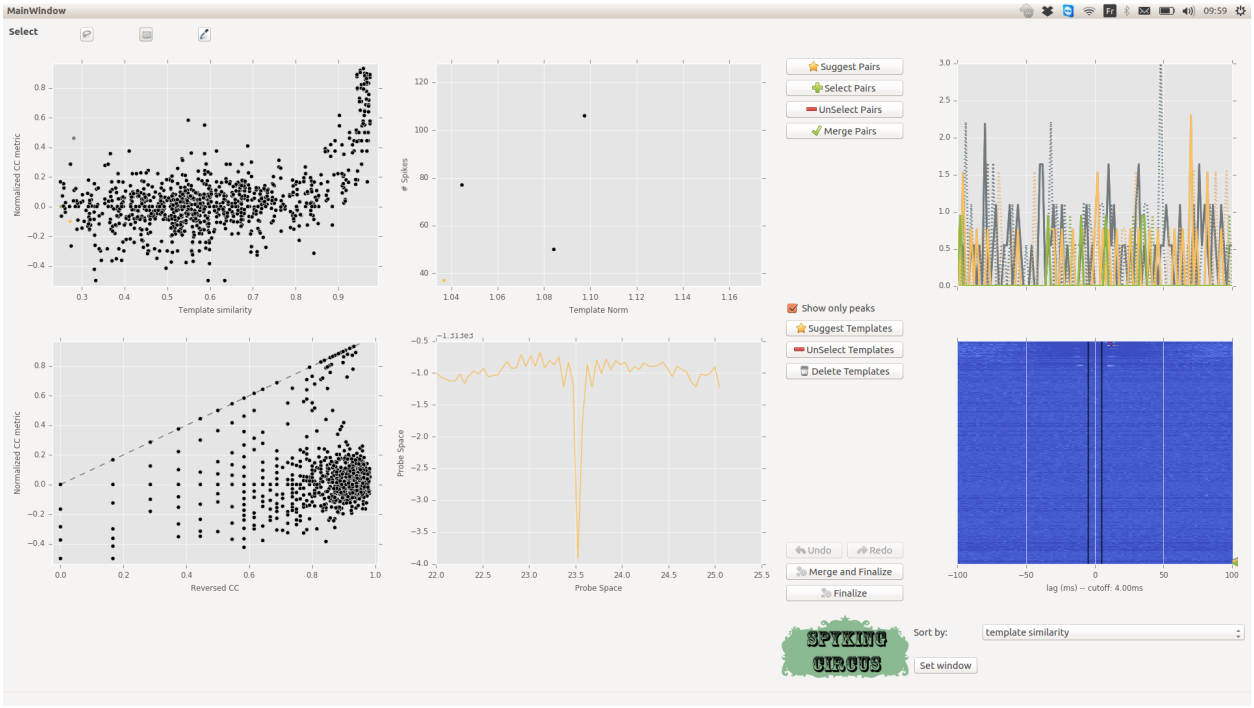


Fig. 2.8: Meta-merging GUI with several templates that are selected

You can then delete those templates, and the GUI will recompute the scores for all pairs.

Saving the results

When you think all merges have been done, you just need to press the `Finalize` Button. This will save everything to file, without overwriting your original results. In fact, it will create new files with the suffix `-merged`, such that you need to use that suffix after if you want to view results in the GUI. Thus, if you want to convert/view those results after, you need to do:

```
>> circus-gui-matlab mydata.extension -e merged
```

CHAPTER 3

Using the GUI

A graphical launcher

For those that do not like the use of a command line, the program now integrates a standalone GUI that can be launched by simply doing:

```
>> spyking-circus-launcher
```

Python GUIs

Preview GUI

In order to be sure that the parameters in configuration file are correct, and before launching the algorithm that will filter the data on-site (and thus mess with them if parameters are wrong), one can use the preview GUI. To do so, simply do:

```
>> spyking-circus path/mydata.extension -p
```

The GUI will display you the electrode mapping, and the first second of the data, filtered, with the detection thresholds as dashed dotted lines. You can then be sure that the value of spike_thresh used in the parameter file is correct for your own data.

Once you are happy with the way data are loaded, you can launch the algorithm.

Note: You can write down the value of the threshold to the configuration file by pressing the button `Write thresh to file`

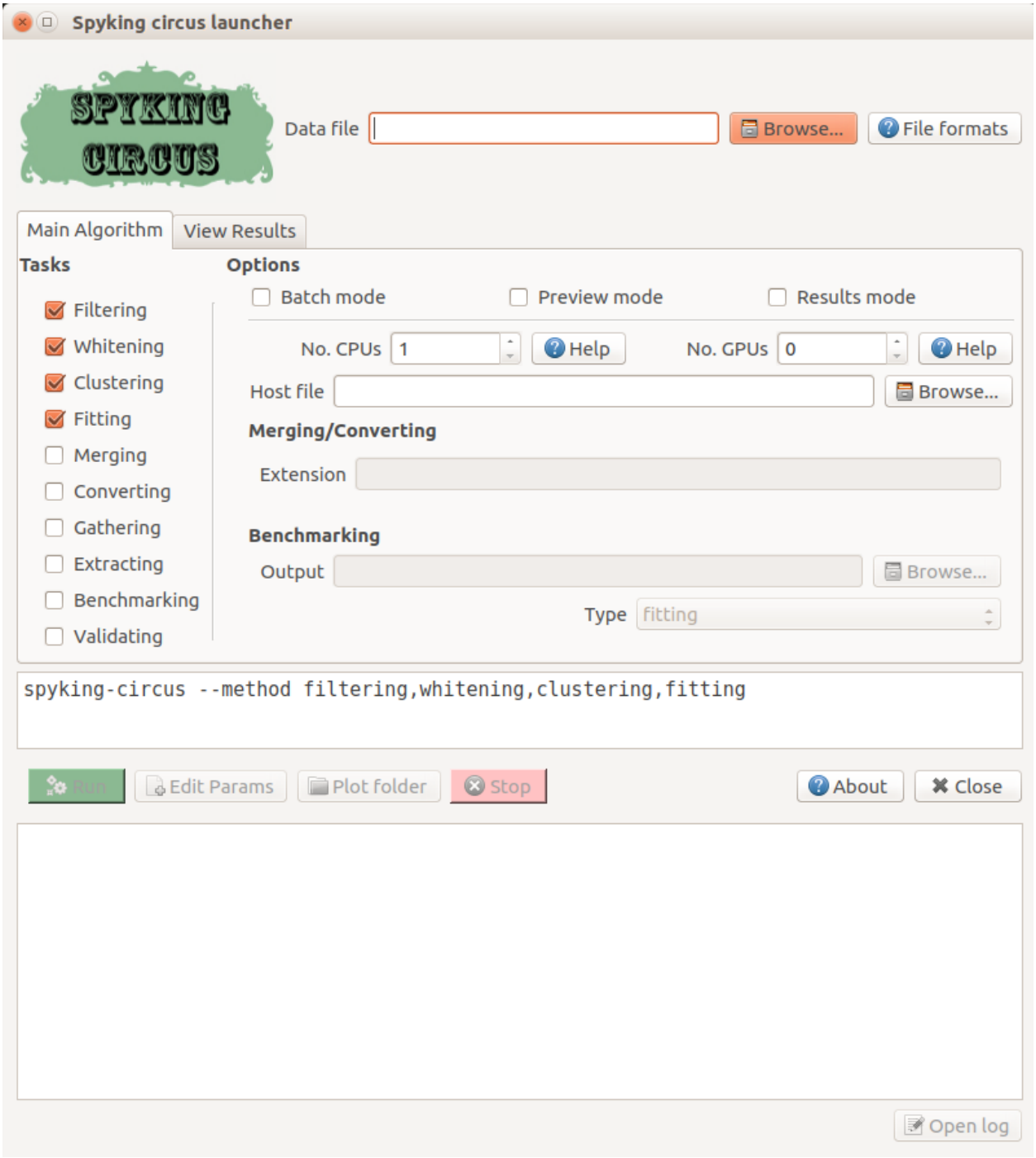


Fig. 3.1: The GUI of the software. All operations described in the documentation can be performed here

Fig. 3.2: A snapshot of the preview GUI. You can click/select one or multiple electrodes, and see the 1s of the activity, filtered, on top with the detection threshold

Result GUI

In order to quickly visualize the results of the algorithm, and get a qualitative feeling of the reconstruction, you can see use a python GUI, similar to the previous one, showing the filtered traces superimposed with the reconstruction provided by the algorithm. To do so, simply do:

```
>> spyking-circus path/mydata.extension -r
```

Warning: If results are not there yet, the GUI will only show you the filtered traces

Note: You can show the residuals, i.e. the differences between the raw data and the reconstruction by ticking the button `Show residuals`

Meta-Merging GUI

See the devoted section on Meta-Merging (see *Automatic Merging*)

Launching the GUIs

You have several options and GUIs to visualize your results, just pick the one you are the most comfortable with!

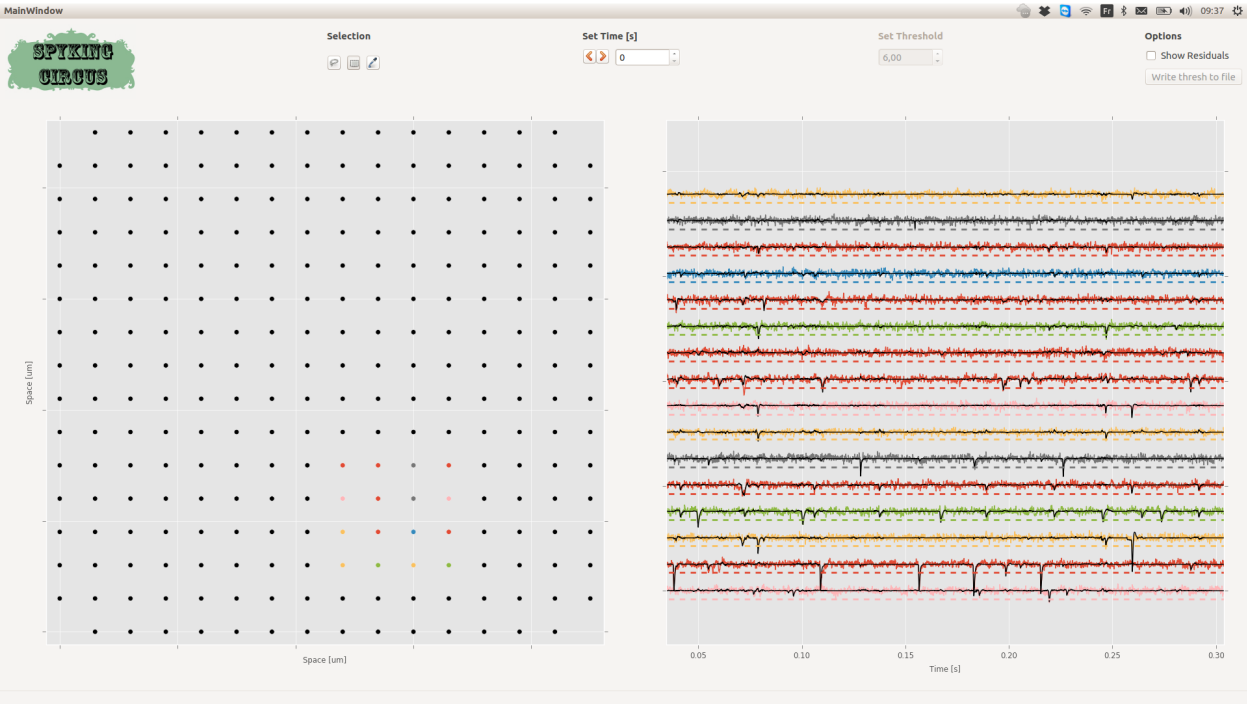


Fig. 3.3: A snapshot of the result GUI. You can click/select one or multiple electrodes, and see the the activity, filtered, on top with the reconstruction provided by the template matching algorithm (in black)

Matlab GUI

To launch the MATLAB GUI provided with the software, you need of course to have a valid installation of MATLAB, and you should be able to simply do:

```
>> circus-gui-matlab path/mydata.extension
```

Note that in a near future, we plan to integrate all the views of the MATLAB GUI into phy

To reload a particular dataset, that have been saved with a special `suffix`, you just need to do:

```
>> circus-gui-matlab path/mydata.extension -e suffix
```

This allows you to load a sorting session that has been saved and not finished. Also, if you want to load the results obtained by the *Meta Merging GUI*, you need to do:

```
>> circus-gui-matlab path/mydata.extension -e merged
```

Python GUI

To launch the Python GUI, you need a valid installation of phy and phycontrib, and you should be able to simply do:

```
>> spyking-circus path/mydata.extension -m converting -c N
```

Followed by:

```
>> circus-gui-python path/mydata.extension
```

As you see, first, you need to export the data to the phy format using the `converting` option (you can use several CPUs with the `-c` flag if you want to export a lot of Principal Components). This is because as long as phy is still under development, this is not the default output of the algorithm. Depending on your parameters, a prompt will ask you if you want to compute all/some/no Principal Components for the GUI. While it may be interesting if you are familiar with classical clustering and PCs, you should not consider exploring PCs for large datasets.

Note: If you want to export the results that you have processed after the *Meta Merging GUI*, you just need to specify the extension to choose for the export:

```
>> spyking-circus path/mydata.extension -m converging -e merged
>> circus-gui-python path/mydata.extension -e merged
```

Panels of the GUIs

In the following, we will mostly talk about the MATLAB GUI, because it is still the default one for the algorithm, but all the concepts are similar across all GUIs.

Warning: The phy GUI is way nicer, but is currently still under active development. We are not responsible for the possible bugs that may be encountered while using it.

Matlab GUI

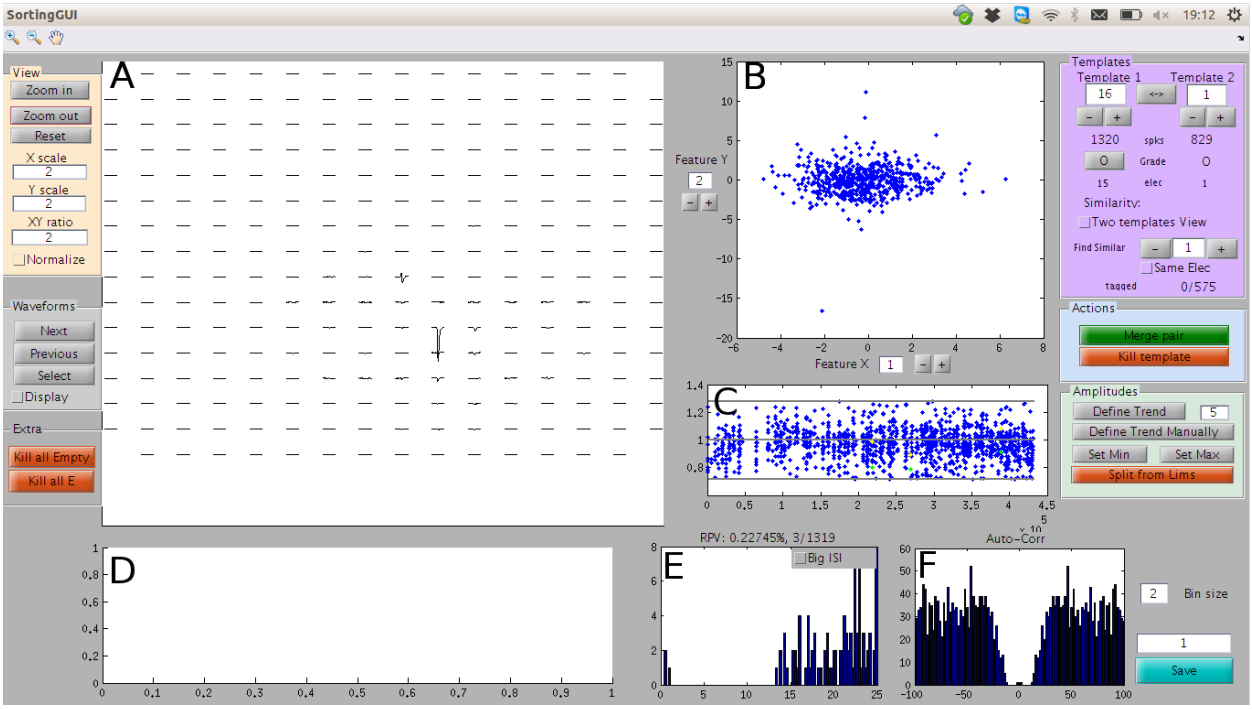


Fig. 3.4: A view of the MATLAB GUI

As you can see, the GUI is divided in several panels:

- **A** A view of the templates
- **B** A view of the features that gave rise to this templates
- **C** A view of the amplitudes over time
- **D** A view for putative repeats, depending on your stimulation
- **E** A view of the Inter Spike Interval Distribution, for that given template
- **F** A view of the Auto/Cross Correlation (Press Show Correlation)

To know more about what to look in those views, see *Basis of Spike Sorting*

Note: At any time, you can save the status of your sorting session, by pressing the `Save` Button. The suffix next to that box will be automatically added to the data, such that you do not erase anything.

To reload a particular dataset, that have been saved with a special `suffix`, you just need to do:

```
>> circus-gui path/mydata.extension -e suffix
```

Python GUI

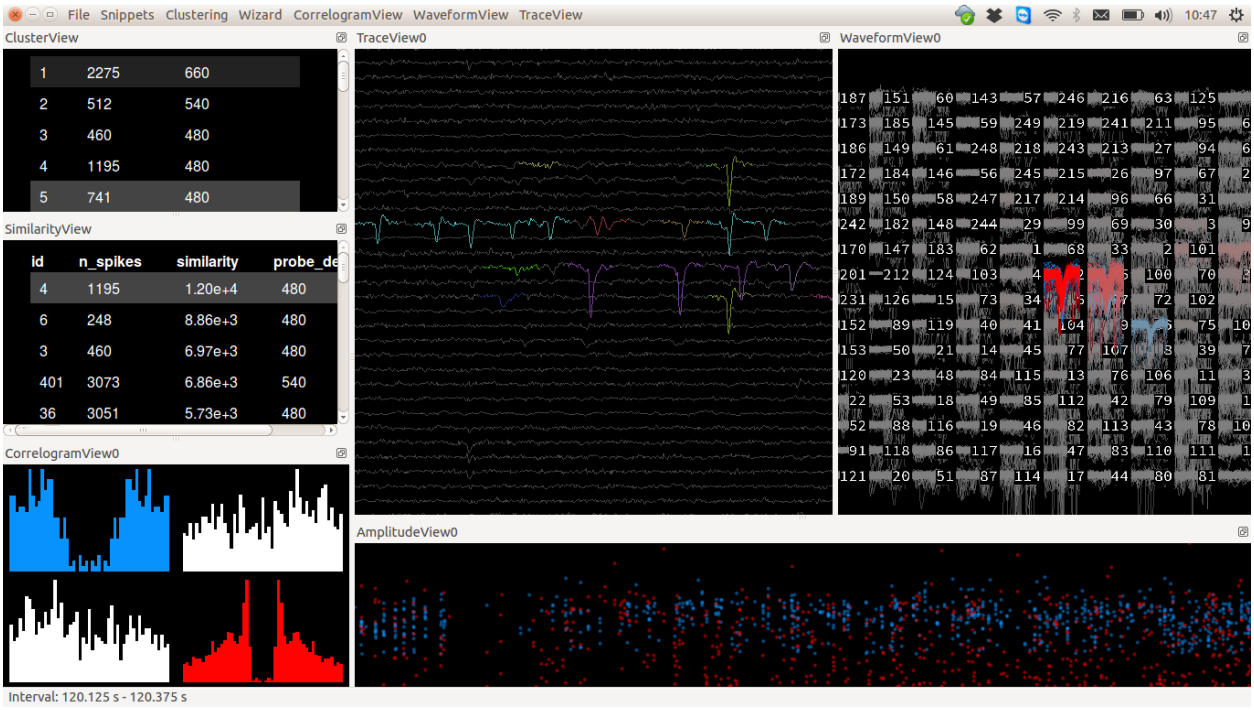


Fig. 3.5: A view of the Python GUI, derived from phy, and oriented toward template matching algorithm. To use it, you need a valid version of phy, and phycontrib

To know more about how to use phy and phycontrib, see the devoted websites. If you want to have a exhaustive description of the sorting workflow performed with phy, please see the phy documentation.

Basis of spike sorting

In this section, we will review the basis of spike sorting, and the key operations that are performed by a human operator, in order to review and assess the quality of the data. The goal here is not to cover all the operations that one need to do when doing spike sorting, but rather to show you how key operations can be performed within the MATLAB GUI. If you want to have a similar description of those steps with phy, please see the phy documentation.

Note: All operations are similar across GUIs, so the key concepts here can be transposed to python/phy GUIs.

Viewing a single template

The algorithm outputs different templates. Each corresponds to the average waveform that a putative cell evokes on the electrodes. The index of the template displayed is on the top right corner. The index can be changed by typing a number on the box or clicking on the plus / minus buttons below it.

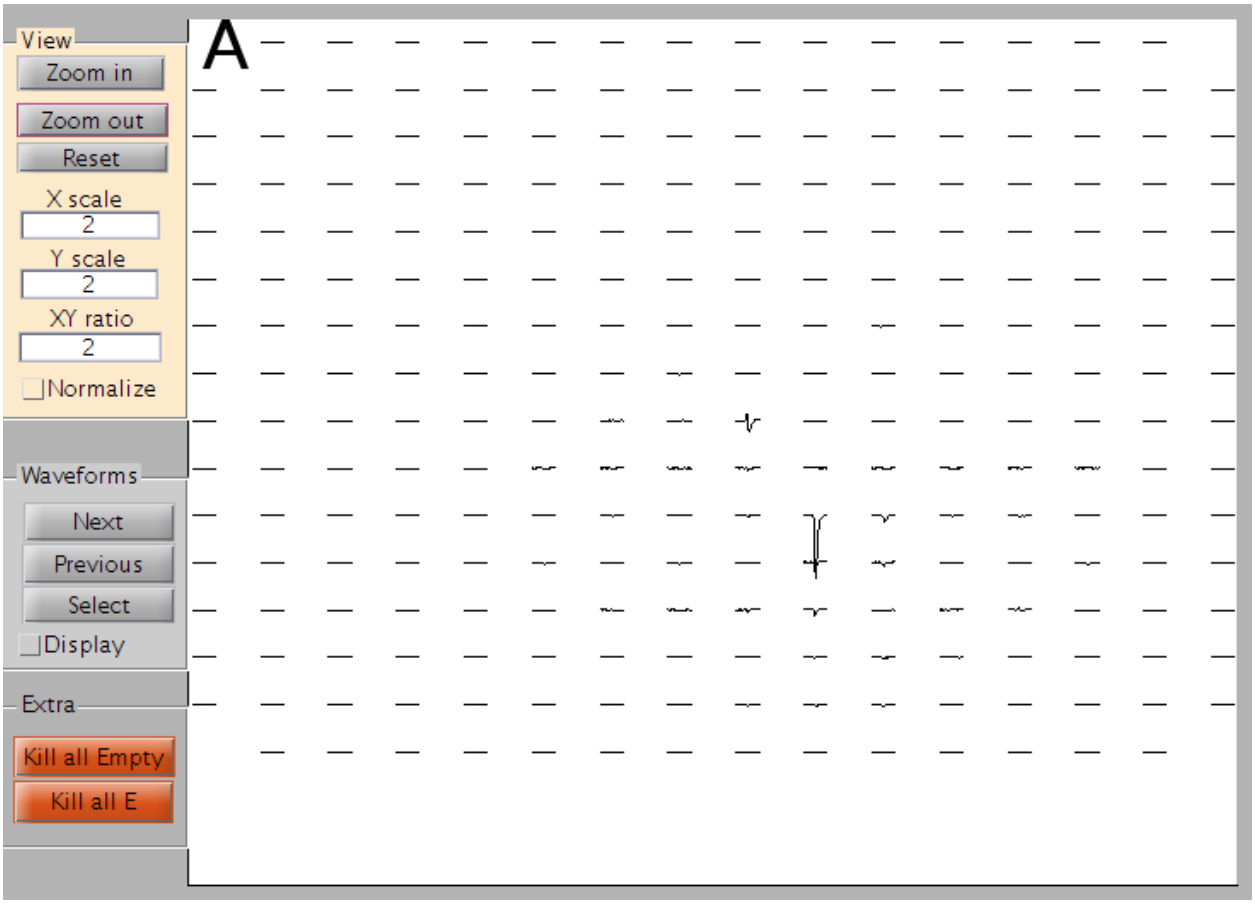


Fig. 3.6: A view of the templates

The large panel A shows the template on every electrode. You can click on the `Zoom in` and `Zoom out` buttons to get a closer look or step back. To adjust the view, you can change the scaling factor for the *X* and *Y* axis by changing the values in the `X scale` and `Y scale` boxes just next to the template view. `Reset` will restore the view to the default view. `Normalize` will automatically adapt the scale to see the most of your template.

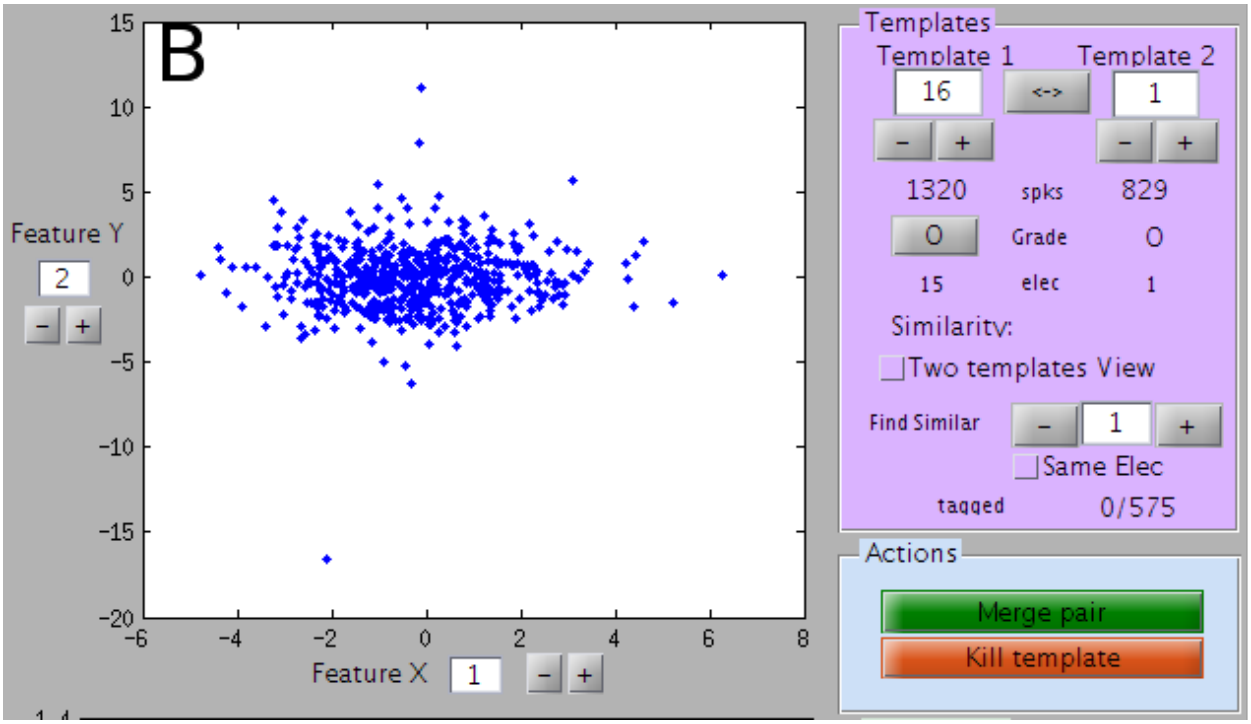


Fig. 3.7: A view of the features

Panel B shows the cluster from which this template has been extracted. Unless you want to redefine the cluster, you don't have to worry about them. You just need to check that the clustering did effectively split clusters. If you see here what you think are two clusters that should have been split, then maybe the parameters of the clustering need to be adjusted (see *documentation on parameters*)

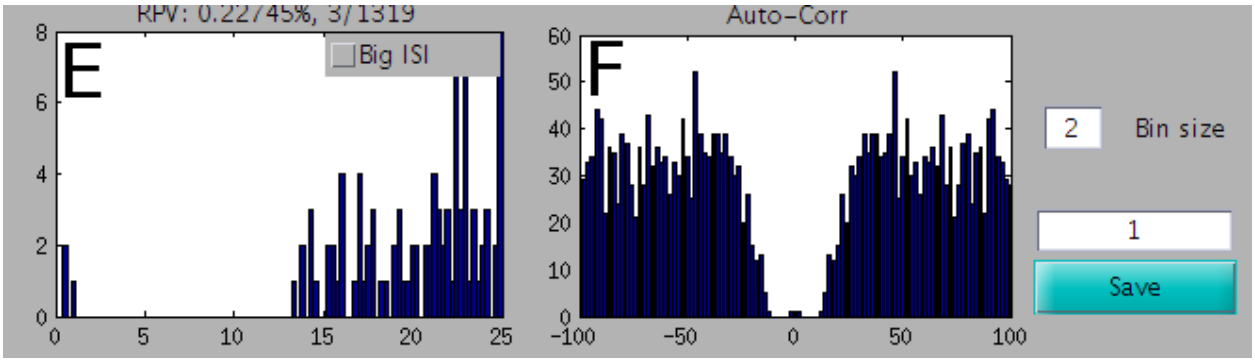


Fig. 3.8: A view of the Inter-Spike Intervals and the AutoCorrelation

Panel E shows the ISI (inter spike interval). You can look at it from 0 to 25 ms, or from 0 to 200 ms if the button `Big ISI` is clicked. Above this panel, the % of refractory period violation is indicated, and a ratio indicates the number of violations / the total number of spikes. Panel F shows the auto-correlation, and you can freely change the time bin.

Note: If you are viewing two templates (see below), then Panel E shows combined ISI for the two templates, and Panel F shows the Cross-Correlogram between the two templates

Cleaning a template

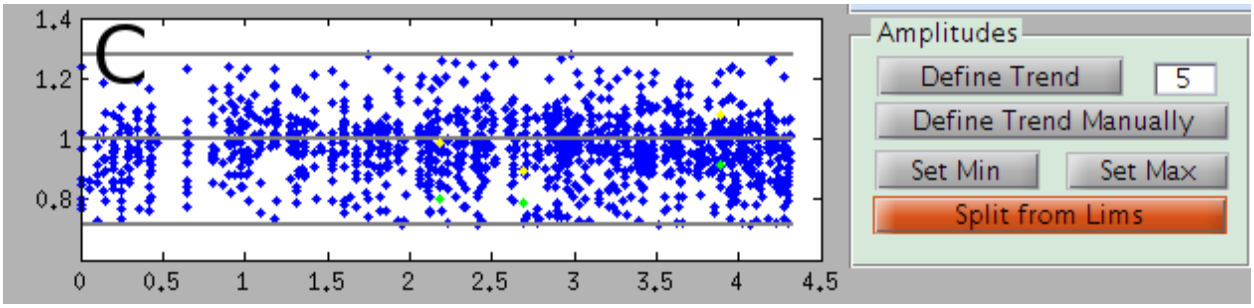


Fig. 3.9: A view of the amplitudes over time

The template is matched all over the data, with a different amplitude each time. Each point of panel C represents a match, the *y*-axis is the amplitude, and *x*-axis the time. When there is a refractory period violation (two spikes too close), the bigger spike appears as a yellow point, and the smaller one in green. The 3 grey lines correspond to the average amplitude, the minimal amplitude and the maximal one.

Many templates should have a large number of amplitudes around 1, as a sanity check that the template matching algorithm is working. However, sometimes, some others can have amplitude that may be anormally small or large. These latter points are usually "wrong matches": they don't correspond to real occurrences of the template. Rather, the algorithm just fitted noise here, or the residual that remains after subtracting templates. Of course, you don't want to consider them as real spikes. So these amplitudes need to be separated from the other ones and removed.

Note: The minimal amplitude is now automatically handled during the fitting procedure, so there should be no need for adjusting the lower amplitude

---

For this purpose, you need to define the limits of the area of good spikes. To define the minimal amplitude, click on the button `Set Min`, and then click on the panel D. The gray line corresponding to the minimal amplitude will be adjusted to pass by the point on which you click. The process holds for `Set Max`.

In some cases, for long recordings where you have a drift, you would like to have an amplitude threshold varying over time. To do so, you need to define first an average amplitude over time. Click on `Define Trend` and see if the grey line follows the average amplitude over time. If not, you can try to modify the number right next to the button: if its value is 10, the whole duration will be divided in 10 intervals, and the median amplitude will be over each of these intervals. Alternatively, you can define this average over time manually by clicking on the `Define Trend Manually` button, then click on all the places by which this trend should pass in panel D, and then press enter.

Once you have set the amplitude min and max correctly, you can split your template in two by clicking on the `Split from Lims` button. The template will be duplicated. One template will only keep the points inside these limits, the other ones will keep the points outside.

Viewing two templates

All these panels can also be used to compare two templates. For this, define the second template in the `Template 2` box (top right), and click on the button `View 2`. This button switches between viewing a single template or viewing two at the same time, in blue and red. In E, you will get the ISI of the merged spike trains, and in F the cross-correlogram between the two cells.

Suggestion of matches

At any time, you can ask the GUI to suggest you the closest template to the one you are currently looking at, by clicking on `Suggest Similar`. By default, the GUI will select the best match among all templates. If the box `Same Elec` is ticked, then the GUI will give you only the best matches on that electrode. You should then be able to see, in the feature space (Panel B), the two distinct clusters. Otherwise, because templates are from point gathered on different electrodes, this comparison does not make sense. If you want to see the *N* - th best match, just enter *N* in the input box next to the `Suggest Similar` Button.

Merging two templates

Very often a single cell is split by the algorithm into different templates. These templates thus need to be merged. When you are looking at one cell, click on the `Suggest similar` button to compare it to templates of similar shape. If the number next to this button, you will compare it to the most similar one, if it is 2, to the second most similar one, and so on. You will be automatically switched to the `View 2` mode (see above). In the middle left, a number between 0 and 1 indicates a coefficient of similarity between the two templates (1=perfect similarity). By ticking the `Normalize` box, the two templates will be normalized to the same maximum.

There are many ways to decide if two templates should be merged or not, but most frequently people look at the cross-correlogram: if this is the same cell, there should be a clear dip in the middle of the cross-correlogram, indicating that two spikes of the two templates cannot be emitted to too close to each other, and thus respecting the refractory period.

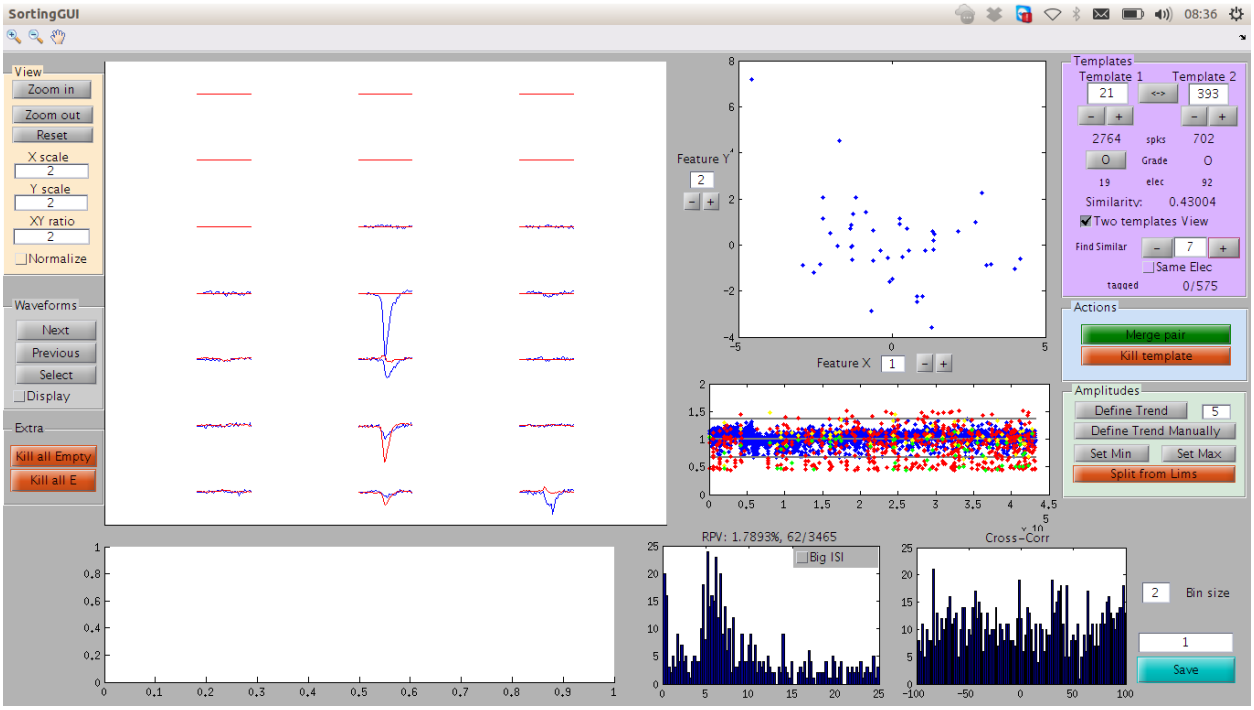


Fig. 3.10: A view of the MATLAB GUI

To merge the two templates together, click on the `Merge` button. The spikes from the two cells will be merged, and only the template of the first one will be kept.

Note that the algorithm is rather on the side of over-dividing the cells into more templates, rather than the opposite, because it is much easier to merge cells than to cluster them further. So you will probably need to do that many times.

Note: Have a look to the Meta Merging GUI, made to perform all obvious merges in your recordings more quickly (see *Automatic Merging*)

Destroying a template

At any time, if you want to throw away a templates, because too noisy, you just need to click on the Button `Kill`. The templates will be destroyed

Warning: There is currently no `Undo` button in the MATLAB GUI. So please consider saving regularly your sorting session, or please consider using phy

Repeats in the stimulation

To display a raster, you need a file containing the beginning and end time of each repeat for each type of stimulus. This file shou

- `rep_begin_time{i}(j)` should contain the start time of the j-th repeat for the i-th type of stimulus.
- `rep_end_time{i}(j)` should contain the end time of the j-th repeat for the i-th type of stimulus.

The times should be specified in sample numbers. These two variables should be stored as a `mat` file in a file called `path/mydata/mydata.stim.mat`, and placed in the same directory than the output files of the algorithm. If available, it will be loaded by the GUI and help you to visualize trial-to-trial responses of a given template.

Give a grade to a cell

Once you have merged a cell and are happy about it, you can give it a grade by clicking on the `O` button. Clicking several times on it will go through different letters from A to E. This extra information can be helpful depending on the analysis you want to perform with your data.

Saving your results

To save the results of your post-processing, click on the `Save` button. A number of files will be saved, with the suffix written in the box right next to the save button. To reload a given spike sorting session, just enter this suffix after the file name when using the `circus-gui-matlab` command (see *documentation on configuration file*):

```
>> circus-gui-matlab mydata.extension -e suffix
```

CHAPTER 4

Advanced Informations

Choosing the parameters

Only few parameters are likely to be modified by the user in the parameter file, depending on the type of data considered. If parameters are not optimal, the code may suggest you to change them. If you want to have a more precise feedback for a given dataset, do not hesitate to ask question to our Google group https://groups.google.com/forum/#!forum/spyking-circus-users, or contact us directly by email.

Note: The longer the recording, the better the code will work. If you have several chunks of recordings, you better concatenate everything into a single large data file, and provide it to the algorithm. This can be done automatically with the `multi-file` mode (see *here*). HOwever, for long recordings, you should turn on the `smart_search` mode (see below).

In vitro

Retina

1. Templates observed are rather large, so `N_t = 5ms` is a decent value. If your final templates are smaller, you should reduce this value, as it reduces the memory usage.
2. A spike can be seen up to 250um away from its initiation site, so this is the default `radius` you should have either in your probe file, either in the parameters
3. Depending on the density of your array, we found that `max_cluster=10` is a decent value. Few electrodes have more than 10 distinct templates

In vivo

Cortex/Hippocampus/Superior Colliculus

1. Templates observed are rather small, so `N_t = 2/3ms` is a decent value. Note that if your templates end up to be smaller, you should reduce this value, as it reduces the memory usage.
2. A spike can be seen up to 100um away from its initiation site, so this is the default `radius` you should have either in your probe file, either in the parameters
3. Depending on the density of your electrodes, we found that `max_cluster=10/15` is a decent value.

Note: If you see too many templates that seems to be mixtures of two templates, this is likely because the automatic merges performed internally are too aggressive. You can change that by playing with the `cc_merge` and `sim_same_elec` parameters (see the *FAQ*)

Low thresholds or long recordings

For long recordings, or if you have low thresholds and a lot of Multi-Unit Activity (MUA), you should consider turning the `smart_search` mode in the `clustering` section to `True`. Such a mode may become the default in future release. Instead of randomly selecting a subset of spikes on all channels, the smart search implements a rejection method algorithm that will try to sample more uniformly all the amplitudes, in order to be sure that all spikes are collected.

Not so dense probes

If you have single channel recordings, or electrodes that are spaced appart by more than 50um, then you should set the `cc_merge` parameter in the `[clustering]` section to 1. Why? Because this parameter will ensure that templates that are scaled copies are not merged automatically. When templates are only over few channels, amplitude is a valuable information that you do not want to discard in order to separate them.

Writing your custom file wrapper

Since 0.5, SpyKING CIRCUS can natively read/write several file formats, in order to ease your sorting workflow. By default, some generic file formats are already implemented (see *the documentation on the file formats*), but you can also write your own wrapper in order to read/write your own custom datafile.

Note that we did not used neo, and we recommend not to do so, because your wrapper should have some functionalities not allo

- it should allow memory mapping, i.e. to read only chunks of your data at a time, slicing either by time or by channels.
- it should read data in their native format, as they will internally be turned into `float32`
- it could allow streaming, if data are internally stored in several chunks

To do so, you simply need to create an object that will inherit from the `DataFile` object described in `circus/files/datafile.py`. The easy thing to understand the structure is to have a look to `circus/files/raw_binary.py` as an example of such a datafile object. If you have questions while writing your wrapper, do not hesitate to be in touch with us.

The speed of the algorithm may slow down a little, depending on your wrapper. For example, currently, we provide an example of a wrapper based on neuroshare (mcd files). This wrapper is working, but slow and inefficient, because the neuroshare API is slow on its own.

Mandatory attributes

Here are the class attributes that you must define:

```
description      = "mydatafile"     # Description of the file format
extension        = [".myextension"] # extensions allowed
parallel_write   = False            # can be written in parallel (using the comm␣
↪object)
is_writable      = False            # can be written
is_streamable    = ['multi-files']  # If the file format can support streams of data [
↪'multi-files' is a default, but can be something else]
_shape           = None             # The total shape of the data (nb time steps, nb␣
↪channels) across streams if any
_t_start         = None             # The global t_start of the data (0 by default)
_t_stop          = None             # The final t_stop of the data, across all␣
↪streams if any
_params          = {}               # The dictionary where all attributes will be␣
↪saved
```

Note that the datafile objects has an internal dictionary `_params` that contains all the values provided by the Configuration Parser, i.e. read from the parameter file in the data section. For a given file format, you can specify:

```
# This is a dictionary of values that need to be provided to the constructor, with␣
↪the corresponding type
_required_fields = {}
```

This is the list of mandatory parameters, along with the type, that have to be specify in the parameter file, because they can not be inferred from the header of your data file. For example:

```
_required_files = {'sampling_rate' : float, 'nb_channels' : int}
```

Then you can also specify some additional parameters, that may have a default value. If they are not provided in the parameter file, this default value is used. For example:

```
# This is a dictionary of values that may have a default value, if not provided to␣
↪the constructor
_default_values  = {'gain' : 1.}
```

At the end, there are 5 mandatory attributes that the code will require for any given file format. Those should be stored in the `_params` dictionary:

- `nb_channels`
- `sampling_rate`
- `data_dtype`
- `dtype_offset`
- `gain`

Custom methods

Here is the list of the function that you should implement in order to have a valid wrapper

Basics IO

You must provide function to open/close the datafile:

```
def _open(self, mode=''):
    '''
        This function should open the file
        - mode can be to read only 'r', or to write 'w'
    '''
    raise NotImplementedError('The open method needs to be implemented for file␣
↪format %s' %self.description)


def _close(self):
    '''
        This function closes the file
    '''
    raise NotImplementedError('The close method needs to be implemented for file␣
↪format %s' %self.description)
```

Reading values from the header

You need to provide a function that will read data from the header of your datafile:

```
def _read_from_header(self):
    '''
        This function is called only if the file is not empty, and should fill the␣
↪values in the constructor
        such as _shape. It returns a dictionary, that will be added to self._params␣
↪based on the constrains given by
        required_fields and default_values
    '''
    raise NotImplementedError('The _read_from_header method needs to be implemented␣
↪for file format %s' %self.description)
```

Such a function must:

- set _shape to (duration, nb_channels)
- set _t_start if not 0
- return a dictionary of parameters that will be used, given the constrains obtained from values in _required_fields and _default_values, to create the DataFile

Reading chunks of data

Then you need to provide a function to load a block of data, with a given size:

```
def read_chunk(self, idx, chunk_size, padding=(0, 0), nodes=None):
    '''
    Assuming the analyze function has been called before, this is the main function
    used by the code, in all steps, to get data chunks. More precisely, assuming your
    dataset can be divided in nb_chunks (see analyze) of temporal size (chunk_size),

        - idx is the index of the chunk you want to load
        - chunk_size is the time of those chunks, in time steps
        - if the data loaded are data[idx:idx+1], padding should add some offsets,
            in time steps, such that we can load data[idx+padding[0]:idx+padding[1]]
        - nodes is a list of nodes, between 0 and nb_channels
    '''
```

```
    raise NotImplementedError('The get_data method needs to be implemented for file␣
↪format %s' %self.description)
```

Note that for convenience, in such a function, you can obtained local t_start, t_stop by using the method `t_start, t_stop = _get_t_start_t_stop(idx, chunk_size, padding)` (see `circus/files/raw_binary.py` for an example). This may be easier to slice your datafile. At the end, data must be returned as `float32`, and to do so, you can also use the internal method `_scale_data_to_float32(local_chunk)`

Writing chunks of data

This method is required only if your file format is allowing write access:

```
def write_chunk(self, time, data):
    '''
        This function writes data at a given time.
        - time is expressed in time step
        - data must be a 2D matrix of size time_length x nb_channels
    '''
    raise NotImplementedError('The set_data method needs to be implemented for file␣
↪format %s' %self.description)
```

Streams

Depending on the complexity of your file format, you can allow several ways of streaming into your data. The way to define streams is rather simple, and by default, all files format can be streamed with a mode called `multi-files`. This is the former `multi-files` mode that we used to have in 0.4 versions (see *multi files*):

```
def set_streams(self, stream_mode):
    '''
        This function is only used for file format supporting streams, and need to␣
↪return a list of datafiles, with
        appropriate t_start for each of them. Note that the results will be using the␣
↪times defined by the streams.
        You can do anything regarding the keyword used for the stream mode, but multi-
↪files is implemented by default
        This will allow every file format to be streamed from multiple sources, and␣
↪processed as a single file.
    '''

    if stream_mode == 'multi-files':
        dirname         = os.path.abspath(os.path.dirname(self.file_name))
        all_files       = os.listdir(dirname)
        fname           = os.path.basename(self.file_name)
        fn, ext         = os.path.splitext(fname)
        head, sep, tail = fn.rpartition('_')
        mindigits       = len(tail)
        basefn, fnum    = head, int(tail)
        fmtstring       = '_%%0%dd%%s' % mindigits
        sources         = []
        to_write        = []
        global_time     = 0
        params          = self.get_description()

        while fname in all_files:
```

```
            new_data    = type(self)(os.path.join(os.path.abspath(dirname), fname),␣
↪params)
            new_data._t_start = global_time
            global_time += new_data.duration
            sources     += [new_data]
            fnum        += 1
            fmtstring    = '_%%0%dd%%s' % mindigits
            fname        = basefn + fmtstring % (fnum, ext)
            to_write    += ['We found the datafile %s with t_start %s and duration %s
↪' %(new_data.file_name, new_data.t_start, new_data.duration)]

        print_and_log(to_write, 'debug', logger)
        return sources
```

Note: When working with streams, you must always defined attributes (such as `t_start`, `duration`, ...) that are local, and defined only for each streams.

As you can see, set_streams is a function that given a `stream_mode`, will read the parameters and return a list of DataFiles, created by slightly changing those parameters. In the case of `multi-files`, this is just a change in the file names, but for some file formats, streams are embedded within the same data structure, and not spread over several files. For example, if you have a look to the file `circus/files/kwd.py` you can see that there is also a mode for streams call `single-file`. If this mode is enabled, the code will process all chunks of data in the HDF5 file, sorted by their keys, as a single giant data file. This is a common situation in experiment. Chunks of data are recorded at several times, but in the same data file. Because they are originating from the same experiment, they better be processed as a whole.

Once those functions are implemented, you simply need to add your wrapper in the list defined in `circus/files/__init__.py`. Or be in touch with us to make it available in the default trunk.

Parallelism

In all your wrappers, if you want to deal with parallelism and do read/write access that will depend on MPI, you have access to an object `comm` which is the MPI communicator. Simply add at the top of your python wrapper:

```
from circus.shared.mpi import comm
```

And then have a look for example `circus/files/hdf5.py` to understand how this is used

Logs

In all your wrappers, if you want to log some informations to the log files (in addition to those logged by default in the DataFile class), you can use the `print_and_log` function. Simply add at the top of your wrapper:

```
from circus.shared.messages import print_and_log
import logging
logger = logging.getLogger(__name__)
```

Then, if you want to log something, the syntax of such a function is:

```
>> print_and_log(list_of_lines, 'debug', logger)
```

Extra steps

The code comes with some additional methods that are not executed by default, but that could still be useful. You can view them by simply doing:

```
>> spyking-circus -h
```

Merging

This option will launh the Meta merging GUI, allowing a fast merging of obvious pairs, based on some automatic computations performed on the cross-correlograms. To launch it, simply use:

```
>> spyking-circus path/mydata.extension -m merging -c N
```

Note: This merging step will not affect your main results, and will generate additional files with the suffix `merged`. You can launch it safely at the end of the fitting procedure, and try various parameters. To know more about how those merges are performed, (see *Automatic Merging*). Note that after, if you want to visualize this `merged` result with the GUIs, you need do use the `-e` parameter, such as for example:

```
>> circus-gui-matlab path/mydata.extension -e merged
```

Gathering

The more important one is the `gathering` option. This option allows you, while the fitting procedure is still running, to collect the data that have already been generated and save them as a temporary result. This methods use the fact that temporal chunks are processed sequentially, so you can, at any time, review what has already been fitted. To do so, simply do:

```
>> spyking-circus path/mydata.extension -m gathering -c N
```

Warning: *N* must be equal to the number of nodes that are currently fitting the data, because you will collect the results from all of them

Note that the data will be saved as if they were the final results, so you can launch the GUI and review the results. If nodes have different speed, you may see gaps in the fitted chunks, because some may be slower than others. The point of this `gathering` function is not to provide you an *exhaustive* view of the data, but simply be sure that everything is working fine.

Converting

As already said in the GUI section, this function allows you to export your results into the phy format. To do so, simply do:

```
>> spyking-circus path/mydata.extension -m converting -c N
```

During the process, you have the option to export or not the Principal Components for all the spikes that have been found, and phy will display them. Note that while this is safe to export all of them for small datasets, this will not scale for very large datasets with millions of spikes.

Warning: For millions of spikes, we do not recommend to export *all* Principal Components. You can export only *some*, but then keep in mind that you can not redefine manually your clusters in phy

Extracting

This option allows the user to get, given a list of spike times and cluster ids, its own templates. For example one could perform the clustering with its own method, and given the results of its algorithms, extract templates and simply launch the template matching part in order to resolve overlapping spikes. To perform such a workflow, you just need to do:

```
>> spyking-circus path/mydata.extension -m extracting,fitting
```

Warning: This option has not yet been tested during the integration in this 0.4 release, so please contact us if you are interested.

Benchmarking

This option allows the user to generate synthetic ground-truth, and assess the performance of the algorithm. We are planning to move it into a proper testsuite, and make its usage more user friendly. Currently, this is a bit undocumented and for internal use only.

In a nutshell, five types of benchmarks can be performed from an already processed file:

- `fitting` The code will select a given template, and inject multiple shuffled copies of it at various rates, at random places
- `clustering` The code will select a given template, and inject multiple shuffled copies of it at various rates and various amplitudes, at random places
- `synchrony` The code will select a given template, and inject multiple shuffled copies of it on the same electrode, with a controlled pairwise correlation coefficient between those cells
- `smart-search` To test the effect of the smart search. 10 cells are injected with various rates, and one has a low rate compared to the others.
- `drifts` Similar to the clustering benchmark, but the amplitudes of the cells are drifting in time, with random slopes

Validating

This method allows to compare the performance of the algorithm to those of a optimized classifier. This is an implementation of the BEER (Best Ellipsoidal Error Rate) estimate, as described in [Harris et al, 2000]. Note that the implementation is slightly more generic, and requires the installation of `sklearn`. To use it, you need to have, if your datafile is `mydata.extension`, a file named `mydata/mydata.npy` which is simply an array of all the ground truth spike times. To know more about the BEER estimate, see the devoted documentation (see *More on the BEER estimate*)

Details of the algorithm

The full details of the algorithm have not been published yet, so we will only draft here the key principles and describe the ideas behind the four key steps of the algorithm. If you can not wait and really would like to know more about all its parameters, please get in touch with pierre.yger@inserm.fr

Note: A full publication showing details/results of the algorithm is available at http://biorxiv.org/content/early/2016/08/04/067843

Filtering

In this first step, nothing incredibly fancy is happening. All the channels are high-pass filtered in order to remove fluctuations, and to do so, we used a classical third order Butterworth filter. This step is required for the algorithm to work.

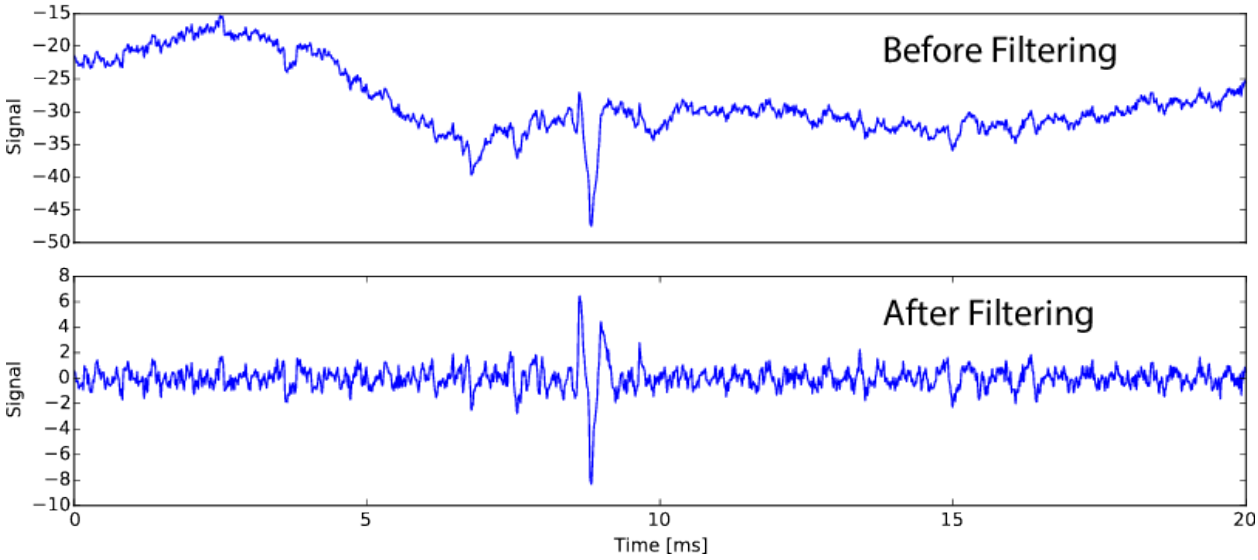


Fig. 4.1: Raw vs. Filtered data

Whitening

In this step, we are removing the spurious spatio-temporal correlations that may exist between all the channels. By detecting temporal periods in the data without any spikes, we compute a spatial matrix and a temporal filter that are whitening the data. This is a key step in most signal processing algorithms.

Warning: Because of this transformation, all the templates and data that are seen after in the MATLAB GUI are in fact seen in this whitened space.

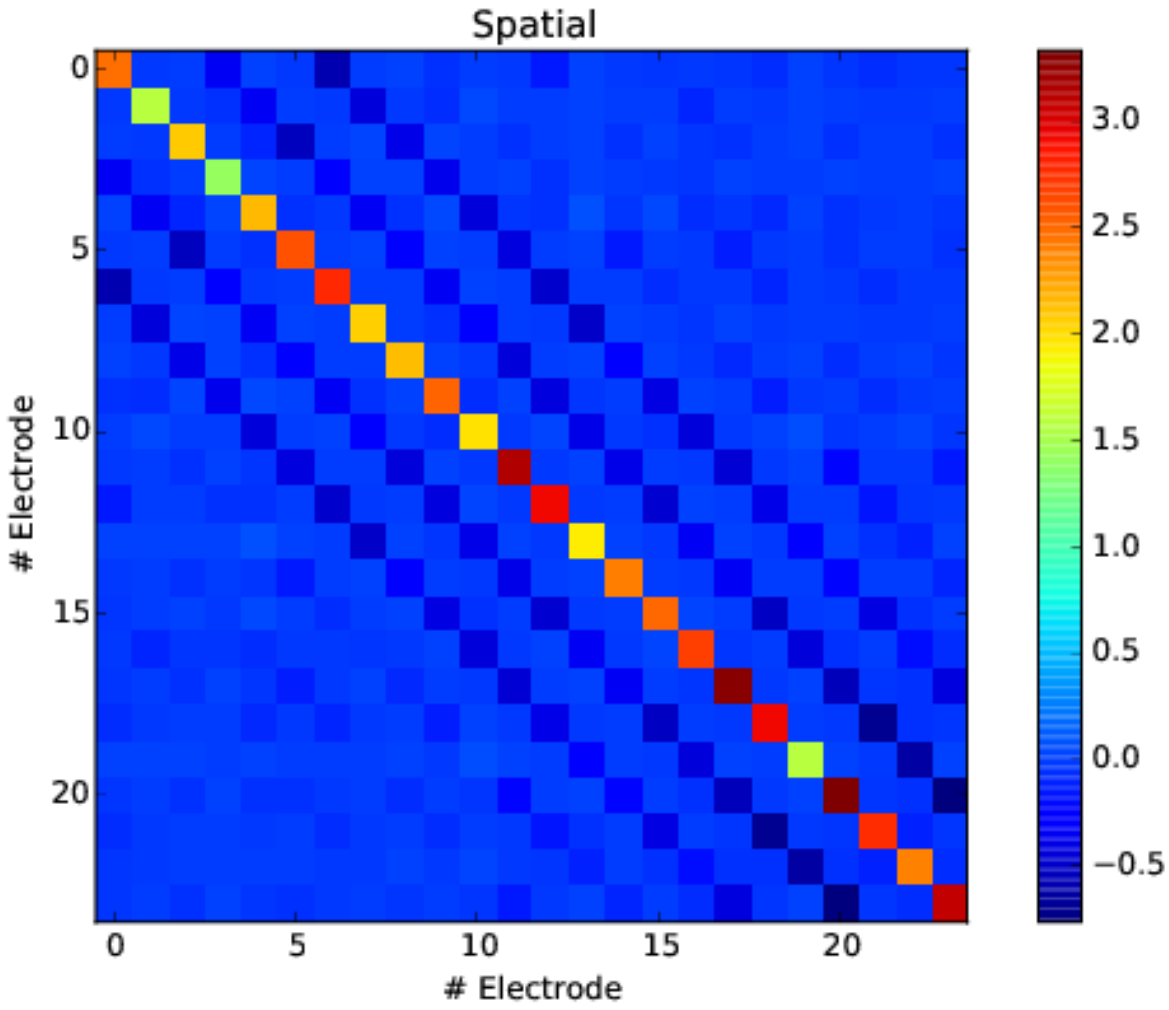


Fig. 4.2: Temporal matrix to perform the whitening of the data for 24 electrodes

Clustering

This is the main step of the algorithm, the one that allows it to perform a good clustering in a high dimensional space, with a smart sub sampling.

A divide and conquer approach

First, we split the problem by pooling spikes per electrodes, such that we can perform N independent clusterings (one per electrode), instead of a giant one. By doing so, the problem becomes intrinsically parallel, and one could easily use MPI to split the load over several nodes.

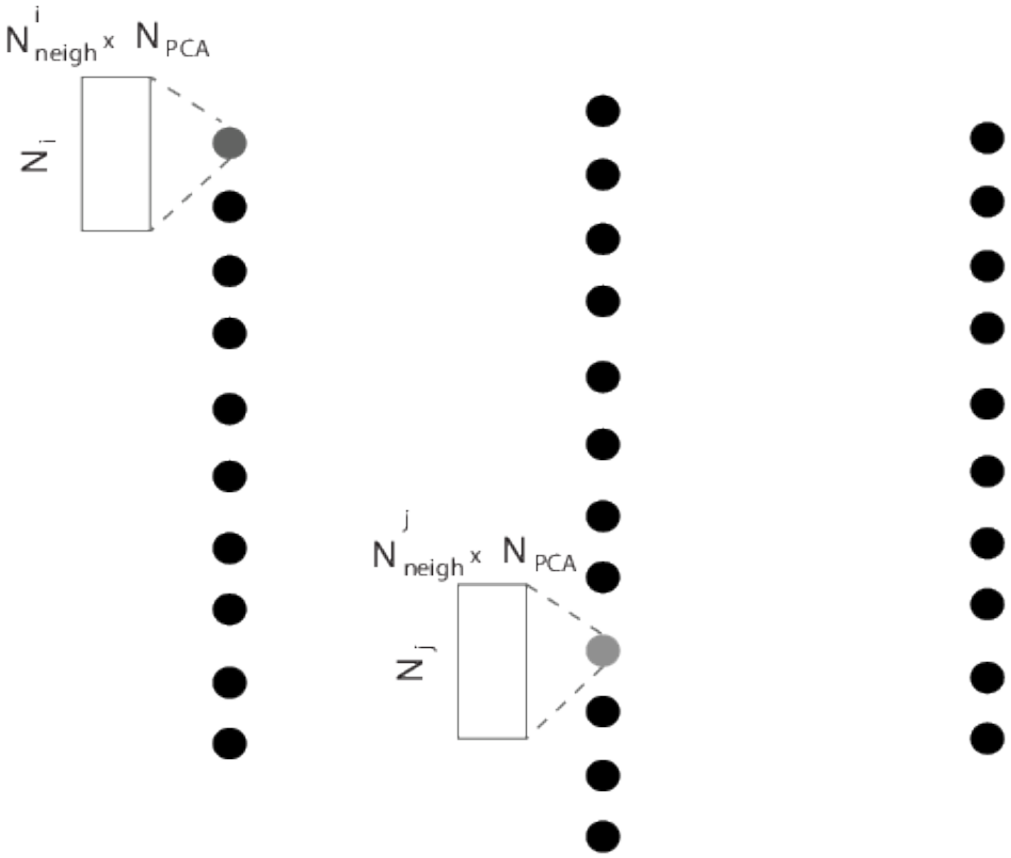


Fig. 4.3: Every spikes is assigned to only one given electrode, such that we can split the clustering problem into N independent clusterings.

A smart and robust clustering

We expanded on recent clustering technique [Rodriguez et Laio, 2014] and designed a fully automated method for clustering the data without being biased by density peaks. In fact, the good point about the template matching approach that we are using is that we just need the *averaged* waveforms, so we don't need to perform a clustering on all the spikes. Therefore, we can cluster only on a subset of all the spikes. They key point is to get a correct subset. Imagine that you have two cells next to the same electrode, but one firing way more than the other. If you are just subsampling

by picking random spikes next to that electrode, you are likely to miss the under-represented neuron. The code is able to solve this issue, and perform what we call a *smart* search of spikes in order to subsample. Details should be published soon.

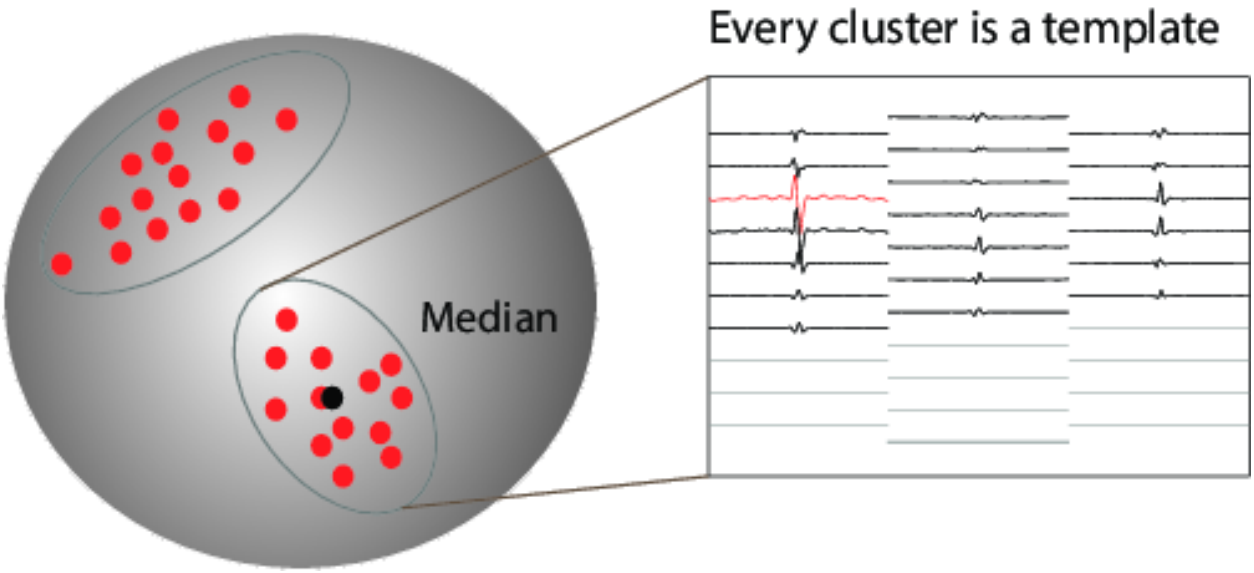


Fig. 4.4: Clustering with smart subsampling in a high dimensional space, leading to spatio-temporal templates for spiking activity triggered on the recording electrodes

Fitting

The fitting procedure is a greedy template matching algorithm, inspired by the following publication [Marre et al, 2012]. The signal is reconstructed as a linear sum of the templates, and therefore, it can solve the problem of overlapping spikes. The good point of such an algorithm is that small temporal chunks can be processed individually (allowing to split the load among several computing units), and that most of the operations performed are matrix operations, thus this can gain a lot from the computing power of modern GPUs.

Generated Files

In this section, we will review the different files that are generated by the algorithm, and at the end of which step. In all the following, we will assume that the data are `path/mydata.extension`. All data are generated in the path `path/mydata/`. To know more about what is performed during the different steps of the algorithm, please see *details on the algorithm*, or wait for the publication.

Whitening

At the end of that step, a single HDF5 file `mydata.basis.hdf5` is produced, containing several objects

- `/thresholds` the *N* thresholds, for all *N* electrodes. Note that values are positive, and should be multiply by the threshold parameter in the configuration file (see *documentation on parameters*)
- `/spatial` The spatial matrix used for whitening the data (size *N* x *N*)
- `/temporal` The temporal filter used for whitening the data (size *Nt* if *Nt* is the temporal width of the template)

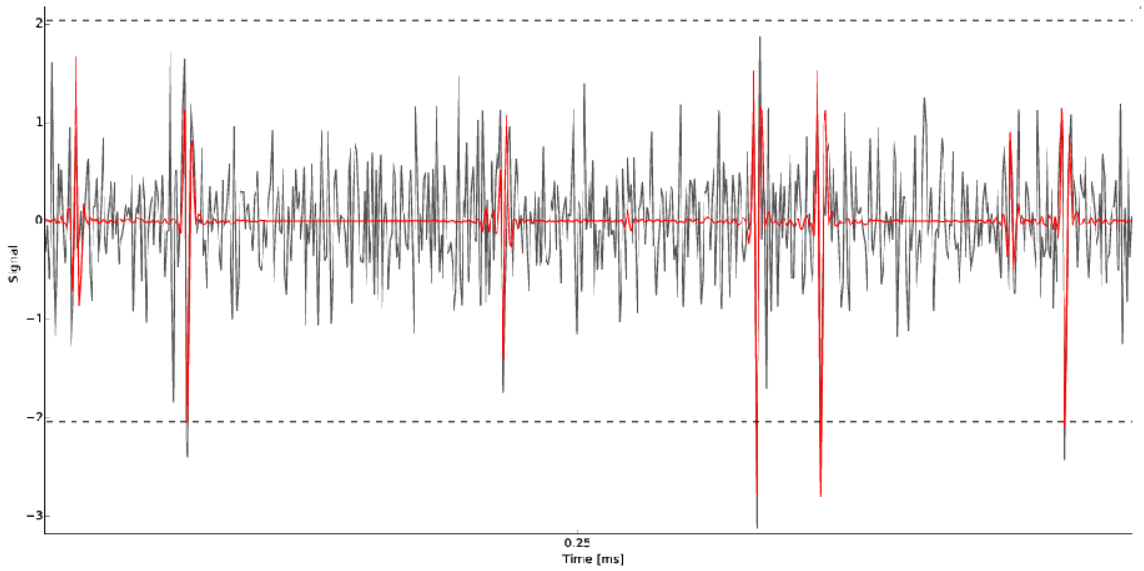


Fig. 4.5: Raw trace on a given electrode and superimposed templates in red. Each time the detection threshold (in dash dotted line) is crossed, the code lookup in the dictionary of template if a match can be found.

- `/proj` and `/rec` The projection matrix obtained by PCA, and also its inverse, to represent a single waveform. (Size *Nt* x *F* if *F* is the number of features kept (5 by default))
- `/waveforms` 1000 randomly chosen waveforms over all channels

Clustering

At the end of that step, several files are produced

- `mydata.clusters.hdf5` A HDF5 file that will encapsulates a lot of informations about the clusters, for every electrodes. What were the points selected, the spike times of those points, what was the labels assigned by the clustering, and also the rho and delta values resulting of the clustering algorithm used [Rodriguez et Laio, 2014]. To be more precise, the file has the following fields
  - `/data_i`: the data points collected on electrode *i*, after PCA
  - `/clusters_i`: the labels of those points after clustering
  - `/times_i`: the spike times at which those spikes are
  - `/debug_i`: a 2D array with rhos and deltas for those points (see clustering algorithm)
  - `/electrodes`: an array with the prefered electrodes of all *K* templates
- `mydata.templates.hdf5` A HDF5 file storing all the templates, and also their orthogonal projections. So this matrix has a size that is twice the number of templates *2k*. Only the first *k* elements are the real templates. Note also that every templates has a given range of allowed amplitudes `limits`, and we are also saving the norms `norms` for internal purposes. To be more precise, the file has the following fields
  - `/temp_shape`: the dimension of the template matrix *N* x *Nt* x *2K* if *N* is the number of electrodes, *Nt* the temporal width of the templates, and *K* the number of templates. Only the first *K* components are real templates

  - `/temp_x`: the x values to reconstruct the sparse matrix
  - `/temp_y`: the y values to reconstruct the sparse matrix
  - `/temp_data`: the values to reconstruct the sparse matrix
  - `/norms` : the *2K* norms of all templates
  - `/limits`: the *K* limits [amin, amax] of the real templates
  - `/maxoverlap`: a *K* x *K* matrix with only the maximum value of the overlaps accross the temporal dimension
  - `/maxlag`: a *K* x *K* matrix with the indices leading to the `maxoverlap` values obtained. In a nutshell, for all pairs of templates, those are the temporal shifts leading to the maximum of the cross-correlation between templates
- `mydata.overlap.hdf5` A HDF5 file used internally during the fitting procedure. This file can be pretty big, and is also saved using a sparse structure. To be more precise, the file has the following fields
  - `/over_shape`: the dimension of the overlap matrix *2K* x *2K* x *2Nt - 1* if *K* is the number of templates, and *Nt* the temporal width of the templates
  - `/over_x`: the x values to reconstruct the sparse matrix
  - `/over_y`: the y values to reconstruct the sparse matrix
  - `/over_data`: the values to reconstruct the sparse matrix

Fitting

At the end of that step, a single HDF5 file `mydata.result.hdf5` is produced, containing several objects

- `/spiketimes/temp_i` for a template *i*, the times at which this particular template has been fitted.
- `/amplitudes/temp_i` for a template *i*, the amplitudes used at the given spike times. Note that those amplitudes has two component, but only the first one is relevant. The second one is the one used for the orthogonal template, and does not need to be analyzed.
- `/gspikes/elec_i` if the `collect_all` mode was activated, then for electrode *i*, the times at which spikes peaking there have not been fitted.

Note: Spike times are saved in time steps

Converting

At the end of that step, several numpy files are produced in a path `path/mydata.GUI`. They are all related to phy, so see the devoted documentation

GUI without SpyKING CIRCUS

MATLAB

You may need to launch the MATLAB GUI on a personal laptop, where the data were not processed by the software itself, so where you only have MATLAB and SpyKING CIRCUS is not installed. This is feasible with the following procedure:

- Copy the the result folder `mydata` on your computer
- Create a MATLAB mapping for the probe you used, i.e. `mapping.hdf5` (see the following procedure below to create it)
- Open MATLAB
- Set the folder `circus/matlab_GUI` as the default path
- Launch the following command `SortingGUI(sampling, 'mydata/mydata', '.mat', 'mapping.hdf5', 2)`

You just need to copy the following code snippet into a file `generate_mapping.py`.

```
import sys, os, numpy, h5py

probe_file = os.path.abspath(sys.argv[1])

def generate_matlab_mapping(probe):
    p         = {}
    positions = []
    nodes     = []
    for key in probe['channel_groups'].keys():
        p.update(probe['channel_groups'][key]['geometry'])
        nodes     +=  probe['channel_groups'][key]['channels']
        positions += [p[channel] for channel in probe['channel_groups'][key]['channels
↪']]
    idx       = numpy.argsort(nodes)
    positions = numpy.array(positions)[idx]

    t     = "mapping.hdf5"
    cfile = h5py.File(t, 'w')
    to_write = {'positions' : positions/10., 'permutation' : numpy.sort(nodes), 'nb_
↪total' : numpy.array([probe['total_nb_channels']])}
    for key in ['positions', 'permutation', 'nb_total']:
        cfile.create_dataset(key, data=to_write[key])
    cfile.close()
    return t

probe = {}
with open(probe_file, 'r') as f:
    probetext = f.read()
    exec probetext in probe

mapping = generate_matlab_mapping(probe)
```

And then simply launch:

```
>> python generate_mapping.py yourprobe.prb
```

Once this is done, you should see a file `mapping.hdf5` in the directory where you launch the command. This is the MATLAB mapping.

Note: If you do not have `h5py` installed on your machine, launch this script on the machine where SpyKING CIRCUS has been launched

phy

After the `converting` step, you must have a folder `mydata/mydata.GUI`. You simply need to copy this folder onto a computer without SpyKING CIRCUS, but only phy. Then you just need to copy the following code snippet into a file `phy_launcher.py`.

```
from phy import add_default_handler
from phy.utils._misc import _read_python
from phy.gui import create_app, run_app
from phycontrib.template import TemplateController

gui_params                    = {}
gui_params['dat_path']        = DATAPATH
gui_params['dir_path']        = DATAPATH/DATAPATH.GUI #If data are left in the␣
↪directory created by SpyKING CIRCUS (where are the .npy)
gui_params['n_channels_dat'] = TOTAL_NB_CHANNELS
gui_params['n_features_per_channel'] = 5
gui_params['dtype']           = DATATYPE
gui_params['offset']          = DATA_OFFSET
gui_params['sample_rate']     = SAMPLE_RATE
gui_params['hp_filtered']     = True

create_app()
controller = TemplateController(**gui_params)
gui = controller.create_gui()

gui.show()
run_app()
gui.close()
del gui
```

You need to edit the appropriate values in capital letters, and then simply copy it into the folder where the raw data are. Now you can do:

```
>> python phy_launcher.py
```

One other option is to simply create a `params.py` file

```
dat_path        = DATAPATH
dir_path        = DATAPATH/DATAPATH.GUI #If data are left in the directory created by␣
↪SpyKING CIRCUS (where are the .npy)
n_channels_dat = TOTAL_NB_CHANNELS
n_features_per_channel = 5
dtype           = DATATYPE
offset          = DATA_OFFSET
sample_rate     = SAMPLE_RATE
hp_filtered     = True
```

Then copy this file into the folder where the raw data are and launch:

```
>> phy template-gui params.py
```

If the raw data are not found, the Traceview will not be displayed. If you really want to see that view, remember that you need to get the raw data **filtered**, so you must also copy them back from your sorting machine.

Example scripts

On this page, you will be very simple example of scripts to load/play a bit with the raw results, either in Python or in Matlab. This is not exhaustive, this is simply an example to show you how you can integrate your own workflow on the results.

> **Warning:** Note that in Python templates (i.e. cells) indices start at 0, while they start at 1 in MATLAB.

Display a template

If you want to display the particular template *i*, as a 2D matrix of size N_e x N_t (respectively the number of channels and the temporal width of your template)

Python

```
from circus.shared.files import *
from pylab import *
params    = load_parameters('yourdatafile.dat')
N_e       = params.getint('data', 'N_e') # The number of channels
N_t       = params.getint('data', 'N_t') # The temporal width of the template
templates = load_data(params, 'templates') # To load the templates
temp_i = templates[:, i].toarray().reshape(N_e, N_t) # To read the template i as a 2D␣
↪matrix
imshow(temp_i, aspect='auto')
```

Matlab

```
tmpfile        = 'yourdata/yourdata.templates.hdf5';
templates_size = double(h5read(tmpfile, '/temp_shape'));
N_e = templates_size(2);
N_t = templates_size(1);
temp_x = double(h5read(tmpfile, '/temp_x') + 1);
temp_y = double(h5read(tmpfile, '/temp_y') + 1);
temp_z = double(h5read(tmpfile, '/temp_data'));
templates = sparse(temp_x, temp_y, temp_z, templates_size(1)*templates_size(2),␣
↪templates_size(3));
templates_size = [templates_size(1) templates_size(2) templates_size(3)/2];
temp_i = full(reshape(templates(:, tmpnum), templates_size(2), templates_size(1)))';
imshow(temp_i)
```

Compute ISI

If you want to compute the inter-spike intervals of cell *i*

Python

```
from circus.shared.files import *
from pylab import *
params    = load_parameters('yourdatafile.dat')
results   = load_data(params, 'results')
spikes    = results['spiketimes']['temp_i']
isis      = numpy.diff(spikes)
hist(isis)
```

Matlab

```
tmpfile = 'yourdata/yourdata.results.hdf5';
spikes  = double(h5read(tmpfile, '/spiketimes/temp_i'));
isis    = diff(spikes);
hist(isis)
```

Display the amplitude over time for a given template

If you want to show a plot of cell *i* spike times vs. amplitudes

Python

```
from circus.shared.files import *
from pylab import *
params    = load_parameters('yourdatafile.dat')
results   = load_data(params, 'results')
spikes    = results['spiketimes']['temp_i']
amps      = results['amplitudes']['temp_i'][:, 0] # The second column are amplitude␣
↪for orthogonal, not needed
plot(spikes, amps, '.')
```

Matlab

```
tmpfile = 'yourdata/yourdata.results.hdf5';
spikes  = double(h5read(tmpfile, '/spiketimes/temp_i'));
amps    = double(h5read(tmpfile, '/amplitudes/temp_i')(:,1));
plot(spikes, amps, '.')
```

Launching the tests

The code has now a dedicated test suite, that will not only test that the code can be launched, but it will also perform some stress tests that will convince you that the code is doing things right. In order to launch the tests, you simply need to do:

```
>> nosetests tests/
```

If you have `nose` installed. You can also only launch some particular tests only:

```
>> nosetests tests/test_complete_workflow.py
```

Note: The test suite is taking some time, because various datasets are generated and processed, so you should not be in a hurry.

What is performed

When you are launching the tests, the code will generate a completely artificial datasets of 5min at 20kHz, composed of some templates with Gaussian noise, on 30 channels. This source dataset is saved in `tests/data/data.dat`.

Note: If you copy your own dataset in `tests/data`, then the tests will use it!

What to see

At the end of every tests, some particular datasets generated using the `benchmarking` mode are stored in `tests/synthetic/`, and plots are generated in `tests/plots/`

BEER estimate

Validating

The code comes with an integrated way to measure the optimal performance of any spike sorting algorithm, given the spike times of a ground truth neuron present in the recording. This can be used by doing:

```
>> spyking-circus mydata.extension -m validating
```

To use it, you need to have, if your datafile is `mydata.extension`, a file named `mydata/mydata.juxta.dat` which is the juxta-cellular signal recorded next to your extracelullar channels. Note that if you have simply the spike times, there is a way to bypass this.

BEER estimate

In a nutshell, to quantify the performance the software with real ground-truth recordings, the code can compute the Best Ellispsiodal Error Rate (BEER), as described in [Harris et al, 2000]. This BEER estimate gives an upper bound on the performance of any clustering-based spike sorting method using elliptical cluster boundaries, such as the one described in our paper. After thresholding and feature extraction, the windowed segments of the trace are labelled according to whether or not they contained a true spike. Half of this labelled data set is then used to train a perceptron whose decision rule is a linear combination of all pairwise products of the features of each segment, and is thus capable of achieving any elliptical decision boundary. This decision boundary is then used to predict the occurrence of spikes in the segments in the remaining half of the labelled data, and the success or failure of these predictions then provide an estimate of the miss and false positive rates.

The code will generate a file `mydata/mydata.beer.dat` storing all the needed information, and will produce several plots.

If you are interested by using such a feature, please contact us!

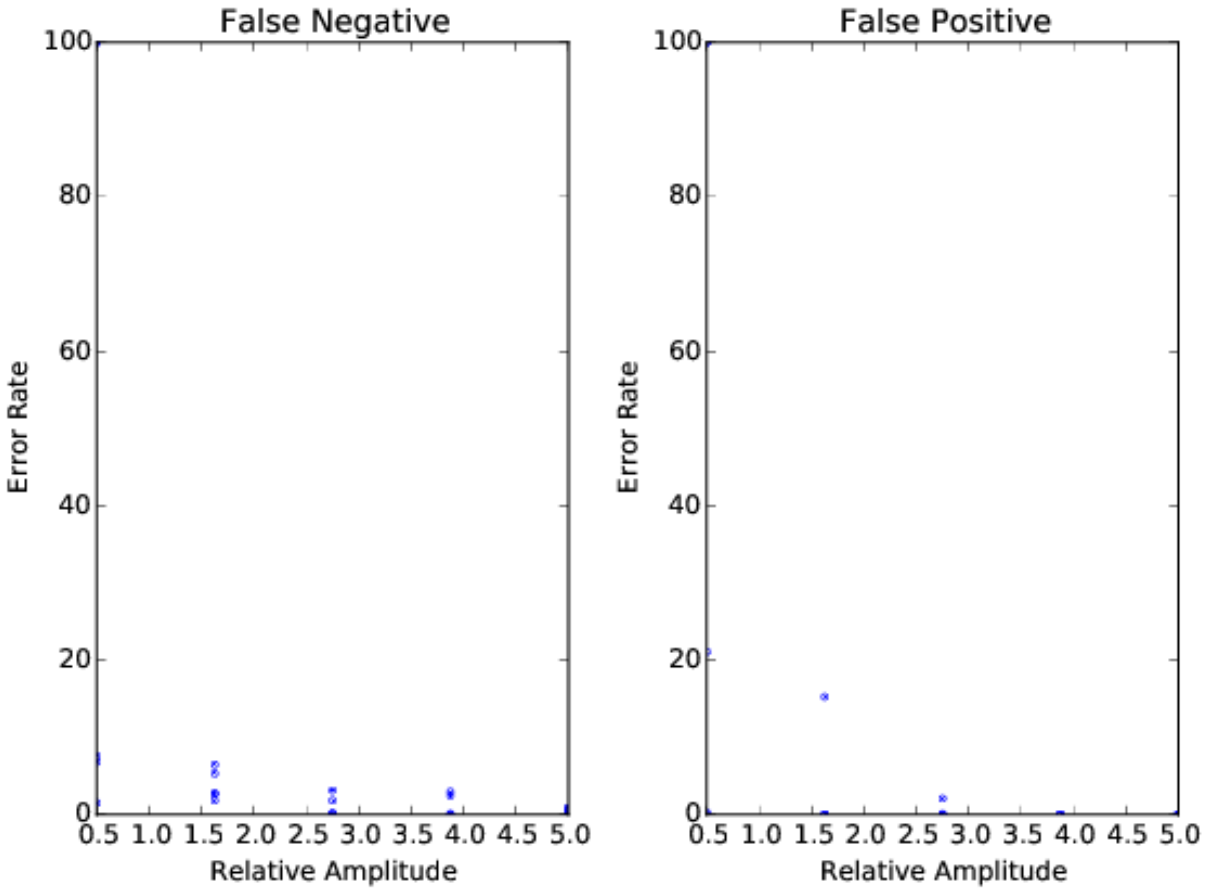


Fig. 4.6: Plots of the tests for the complete workflow. 25 templates at various rates/amplitudes are injected into the source datasets, and performance are shown here.

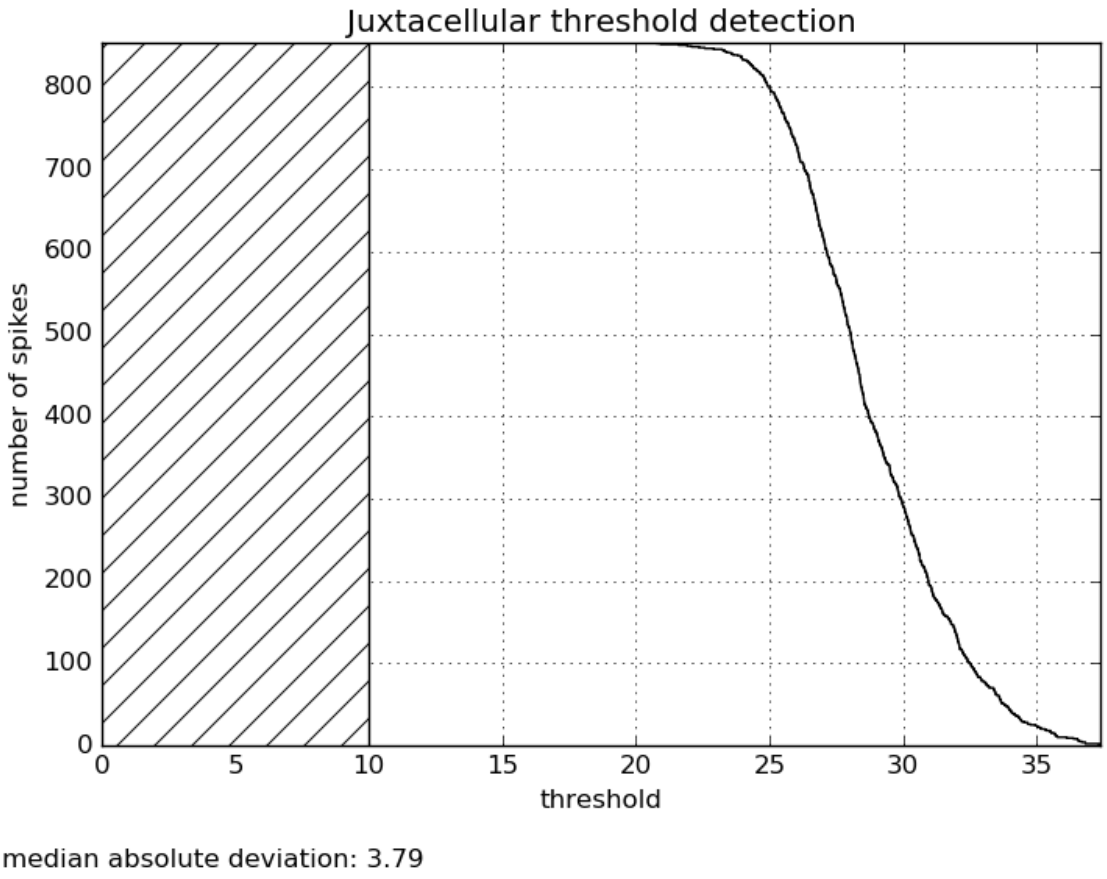


Fig. 4.7: Distribution of the number of juxta-cellular spikes as function of the detection thresholds (to know where it has to be defined)

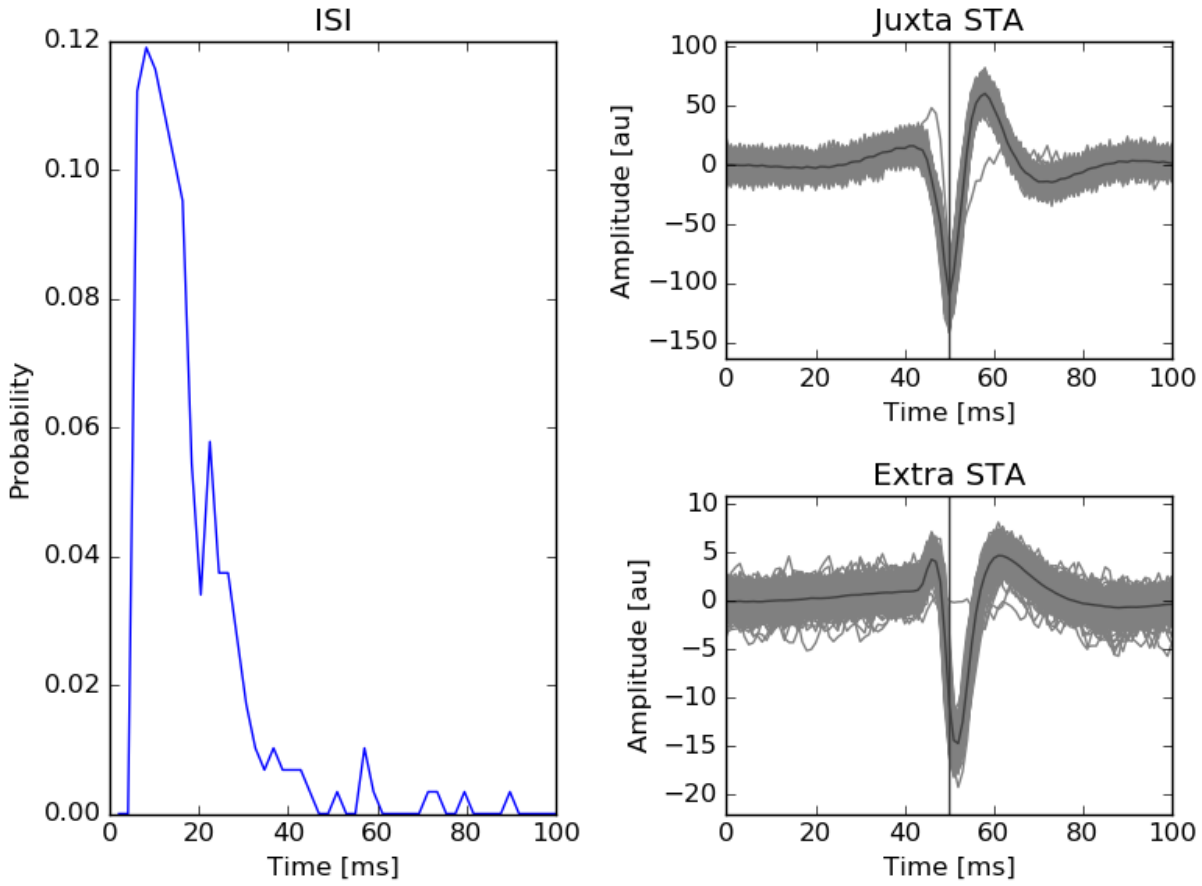


Fig. 4.8: ISI and mean waveforms triggered by the juxta-cellular spikes

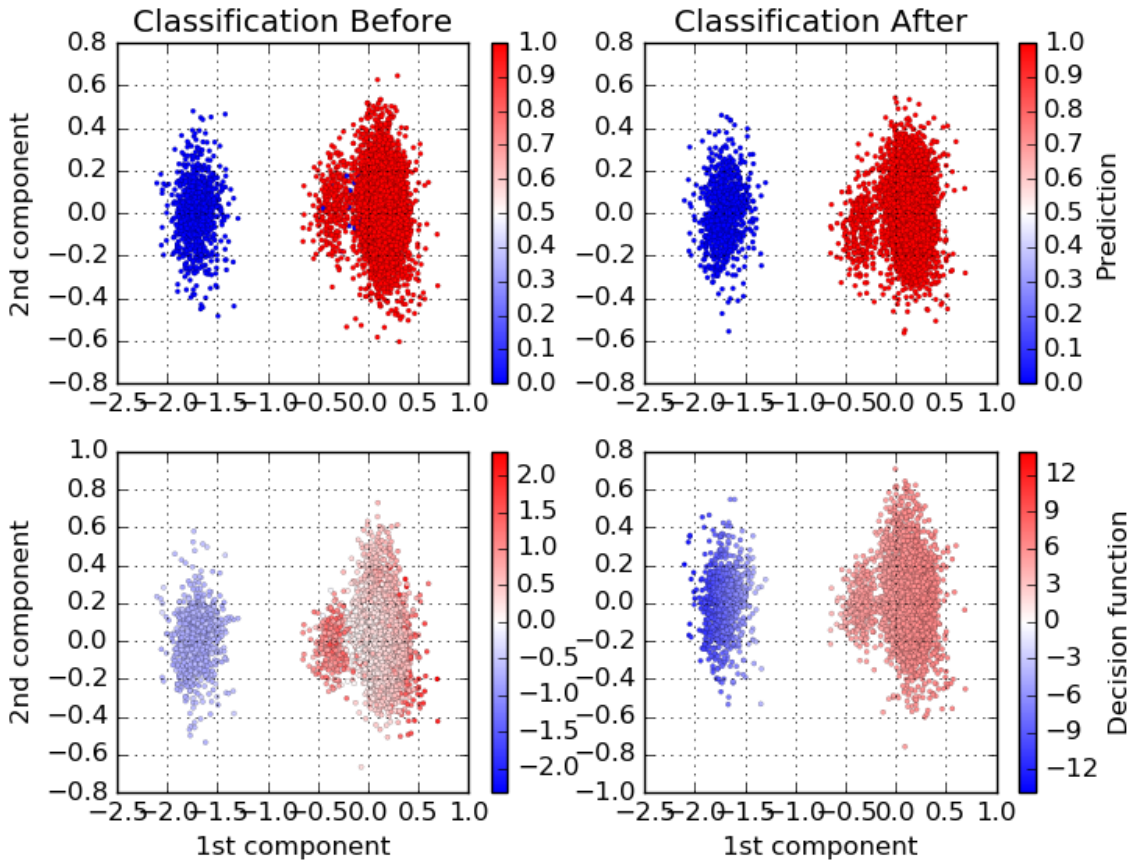


Fig. 4.9: Decision boundaries of the BEER classifier before and after learning.

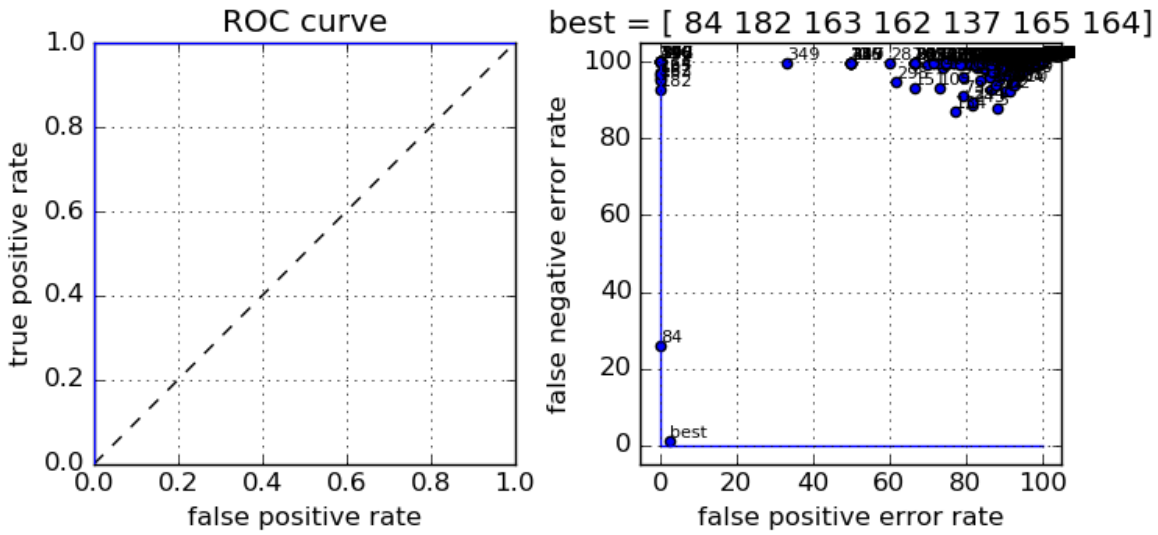


Fig. 4.10: The complete ROC curve for the classifier, and all the templates found by the algorithm, superimposed.

CHAPTER 5

Known issues

In this section, you will find all information you need about possible bugs/comments we got from users. The most common questions are listed in the FAQ, or you may have a look to more specialized sections

Frequently Asked Questions

Here are some questions that are popping up regularly. You can ask some or get answers on our Google Group https://groups.google.com/forum/#!forum/spyking-circus-users

- **I can not install the software**

Note: Be sure to have the latest version from the git folder. We are doing our best to improve the packaging and be sure that the code is working on all platforms, but please be in touch we us if you encounter any issues

---

- **Is is working with Python 3?**

Note: Yes, the code is compatible with Python 3

- **The data displayed do not make any sense**

Note: Are you sure that the data are properly loaded? (see `data` section of the parameter file, especially `data_dtype`, `data_header`). Test everything with the preview mode by doing:

```
>> spyking-circus mydata.extension -p
```

- **Can I process single channel datasets, or coming from not so-dense electrodes?**

Note: Yes, the code can handle spikes that will occur only on a single channel, and not on a large subset. However, you may want to set the `cc_merge` parameter in the `[clustering]` section to 1, to prevent any global merges. Those global merges are indeed performed automatically by the algorithm, before the fitting phase. It assumes that templates that are similar, up to a scaling factor, can be merged because they are likely to reflect bursting neurons. But for few channels, where spatial information can not really be used to disentangle templates, the amplitude is a key factor that you want to keep. Also, you may need to turn on the `smart_search` mode in the `clustering` section, because as you have few channels, you want to collect spikes efficiently.

- **Something is wrong with the filtering**

Note: Be sure to check that you are not messing around with the `filter_done` flag, that should be automatically changed when you perform the filtering. You can read the troubleshooting section on the filtering *here*

- **I see too many clusters, at the end, that should have been split**

Note: The main parameters that you can change will be `cc_merge` and `sim_same_elec` in the `[clustering]` section. They are controlling the number of *local* (i.e. per electrode) and *global* (i.e. across the whole probe layout) merges of templates that are performed before the fitting procedure is launched. By reducing `sim_same_elec` (can not be less than 0), you reduce the *local* merges, and by increasing `cc_merge` (can not be more than 1), you reduce the *global* merges. A first recommendation would be to set `cc_merge` to 1. You might also want to turn on the `smart_search` parameter in the `clustering` section. This will force a smarter collection of the spikes, based on rejection methods, and thus should improve the quality of the clustering.

- **Do I really need to have a GPU?**

Note: In 0.4, it is likely than using 1 GPU is faster than 1 CPU. However, using several CPU is faster than a single GPU, so we recommend using a lot of CPU if you want to gain speed. Dedicated CUDA kernels should be there for 0.5, bringing back the full power of GPUs.

- **Memory usage is saturating for thousands of channels**

Note: If you have a very large number of channels (>1000), then the default size of 60s for all the data blocks loaded into memory during the different steps of the algorithm may be too big. In the `whitening` section, you can at least change it by setting `chunk_size` to a smaller value (for example 10s), but this may not be enough. If you want the code to always load smaller blocks during all steps of the algorithm `clustering, filtering`, then you need to add this `chunk_size` parameter into the `data` section.

- **How do I read the templates in Python?**

Note: Templates are saved as a sparse matrix, but you can easily get access to them. For example if you want to read the template *i*, you have to do

```
from circus.shared.files import *
params    = load_parameters('yourdatafile.dat')
N_e       = params.getint('data', 'N_e') # The number of channels
N_t       = params.getint('data', 'N_t') # The temporal width of the template
```

```
templates = load_data(params, 'templates') # To load the templates
temp_i = templates[:, i].toarray().reshape(N_e, N_t) # To read the template i as a 2D␣
↪matrix
```

To know more about how to play with the data, and build your own analysis, either in Python or MATLAB you can go to our *dedicated section on analysis*

- **After merging templates with the Meta Merging GUI, waveforms are not aligned**

Note: By default, the merges do not correct for the temporal lag that may exist between two templates. For example, if you are detecting both positive and negative peaks in your recordings, you may end up with time shifted copies of the same template. This is because if the template is large enough, crossing both positive and negative thresholds at the same time, the code will collect positive and negative spikes, leading to twice the same template, misaligned. We are doing our best, at the end of the clustering step, to automatically merge those duplicates based on the cross-correlation (see parameter `cc_merge`). However, if the lag between the two extrema is too large, or if they are slightly different, the templates may not be fused. This situation will bring a graphical issue in the phy GUI, while reviewing the result: if the user decided in the Meta Merging GUI to merge the templates, the waveforms will not be properly aligned. To deal with that, you simply must to set the `correct_lag` parameter in the `[merging]` section to `True`. Note that such a correction can not be done for merges performed in phy.

Filtering

The filtering is performed once, on the data, without any copy. This has pros and cons. The pros is that this allow the code to be faster, avoiding filtering on-the-fly the data each time temporal chunks are loaded. The cons is that the user has to be careful about how this filtering is done.

Wrong parameters

If you filled the parameter files with incorrect values either for the data type, header, or even the number of channels (i.e. with a wrong probe file), then the filtering is likely to output wrong data in the file itself. If you are facing issues with the code, always be sure that the informations displayed by the algorithm before any operations are correct, and that the data are correctly read. To be sure, use the preview GUI before launching the whole algorithm (see *Python GUI*):

```
>> spyking-circus mydata.extension -p
```

Interruption of the filtering

The filtering is performed in parallel by several nodes, each of them in charge of a subset of all the temporal chunks. This means that if any of them is failing because of a crash, or if the filtering is interupted by any means, then you have to copy again the entire raw file and start again. Otherwise, you are likely to filter twice some subparts of the data, leading to wrong results

Flag filter_done

To let the code know that the filtering has been performed, you can notice at the bottom of the configuration file a flag `filter_done` that is False by default, but that becomes `True` only after the filtering has been performed. As long as this parameter files is ketp along with your data, the algorithm, if relaunched, will not refilter the file.

Warning: If you delete the configuration file, but want to keep the same filtered data, then think about setting this flag manually to `True`

Whitening

No silences are detected

This section should be pretty robust, and the only error that you could get is a message saying that no silence were detected. If this is the case, this is likely that the parameters are wrong, and that the data are not properly understood. Be sure that your data are properly loaded by using the preview mode:

```
>> spyking-circus mydata.extension -p
```

If this is the case, please try to reduce the `safety_time` value. If no silences are detected, then your data may not be properly loaded.

Whitening is disabled because of NaNs

Again, this should be rare, and if this warning happens, you may try to get rid of this warning by changing the parameters of the `whitening` section. Try for example to increase `safety_time` for example to `3`, or try to change the value of `chunk_size`. We may enhance the robustness of the whitening in future releases.

CHAPTER 6

Citations

How to cite SpyKING CIRCUS

Note: If you are using SpyKING CIRCUS for your project, please cite us

- Yger P., Spampinato, G.L.B, Esposito E., Lefebvre B., Deny S., Gardella C., Stimberg M., Jetter F., Zeck G. Picaud S., Duebel J., Marre O., *Fast and accurate spike sorting in vitro and in vivo for up to thousands of electrodes*, bioRxiv, 67843, 2016

Selected publications using SpyKING CIRCUS

Here is a non exhaustive list of papers using SpyKING CIRCUS. Do not hesitate to send us a mail in order to add a paper

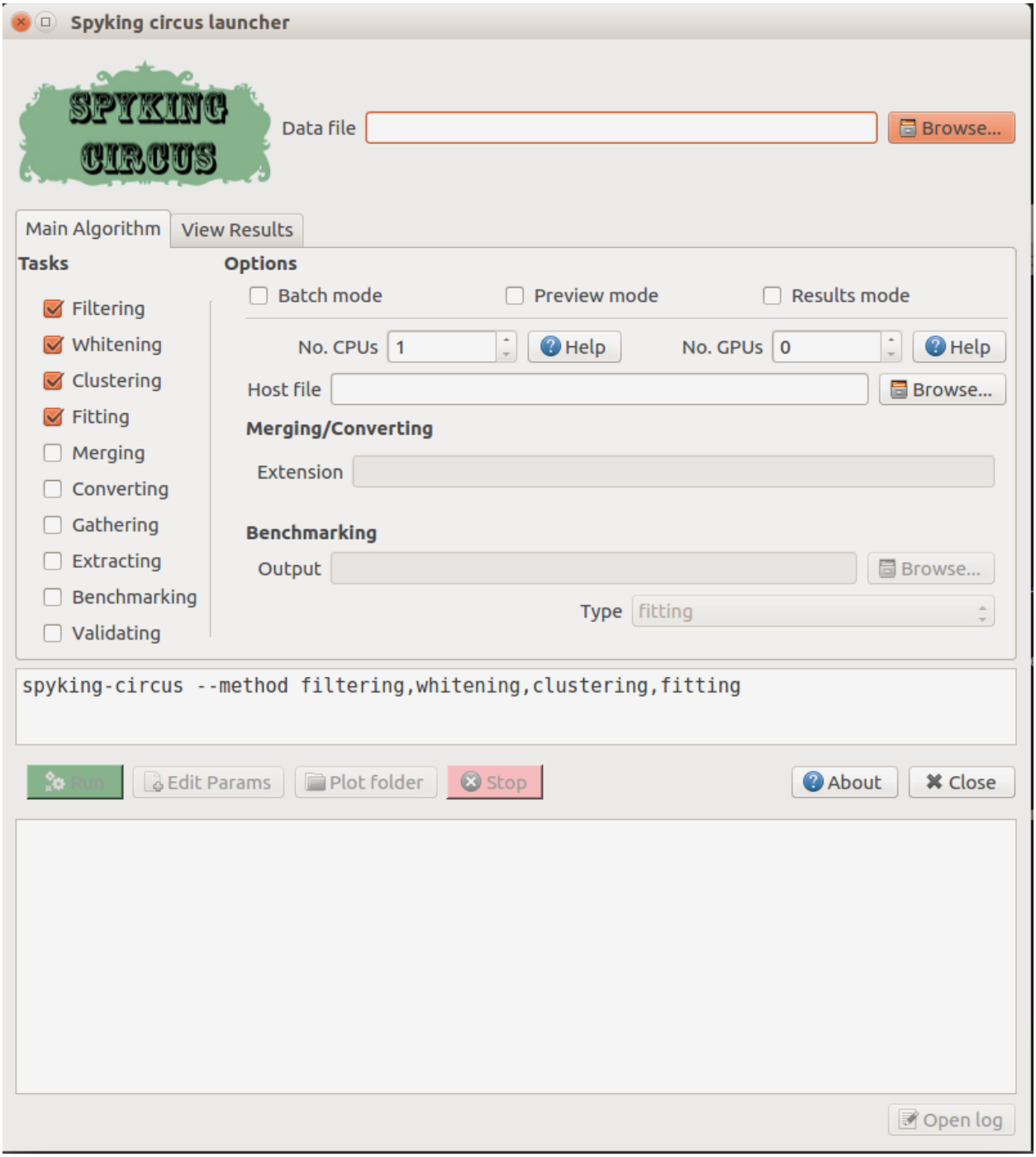
2017

- Wilson C.D., Serrano G. O., Koulakov A. A., Rinberg D., *Concentration invariant odor coding*, bioRxiv, 125039, 2017
- Ferrari U., Gardella C., Marre O., Mora T., *Closed-loop estimation of retinal network sensitivity reveals signature of efficient coding*, bioRxiv, 096313, 2017

2016

- Denman D.J, Siegle J.H., Koch C., Reid R.C., Blanche T.J., *Spatial organization of chromatic pathways in the mouse dorsal lateral geniculate nucleus*, Journal of Neuroscience, 2016

- Yger P., Spampinato, G.L.B, Esposito E., Lefebvre B., Deny S., Gardella C., Stimberg M., Jetter F., Zeck G. Picaud S., Duebel J., Marre O., *Fast and accurate spike sorting in vitro and in vivo for up to thousands of electrodes*, bioRxiv, 67843, 2016


Spyking circus launcher
SPYKING CIRCUS
Data file
Browse...
Main Algorithm
View Results
Tasks
Filtering
Whitening
Clustering
Fitting
Merging
Converting
Gathering
Extracting
Benchmarking
Validating
Options
Batch mode
Preview mode
Results mode
No. CPUs
1
Help
No. GPUs
0
Help
Host file
Browse...
Merging/Converting
Extension
Benchmarking
Output
Browse...
Type
fitting
spyking-circus --method filtering,whitening,clustering,fitting
Run
Edit Params
Plot folder
Stop
About
Close
Open log