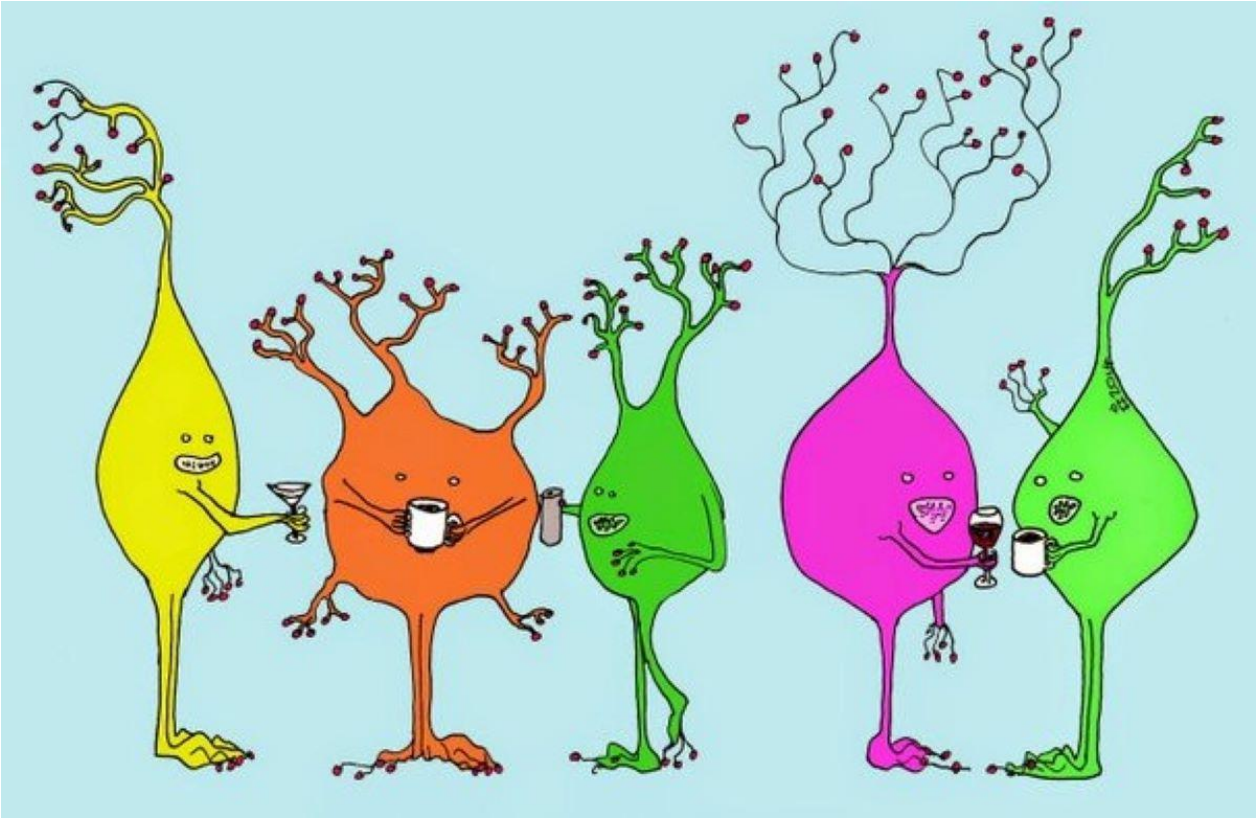
Processing Axona and OpenEphys data



Collection of scripts and procedures for processing Axona and OpenEphys data recorded in the Jezek lab. Does not include instructions for sgClust.

Also updated on www.github.com/SusanL82

By Susan Leemburg (susan.leemburg@lfp.cuni.cz or susanleemburg@gmail.com)

Contents

Installation (for using Klustakwik)

Spyder and Anaconda navigator

I used Spyder for most procedures in this document. As someone who is more familiar with Matlab, I find Spyder the most convenient way to interface with Python.
It should be possible to run all python scripts in Jupyter notebooks as well. It is currently not possible to use my code as functions from the command prompt. I do not know how to convert them for this use yet, but it should be possible and I'll probably make this update in the future.

Installing SpikeInterface with klustakwik

Klustakwik and klusta are old legacy software require the following installations of Python and SpikeInterface.

1. **Create a conda environment with Python 3.7**
   I used the anaconda navigator for this and created the environment *klusta.* You can name your environment whatever you like.

2. **Install klustakwik**
   In the conda command prompt, activate your environment:
   ```
   conda activate klusta
   ```

   Then install klustakwik and its prerequisites with pip. Type:
   ```
   pip install Cython h5py tqdm
   pip install click klusta klustakwik2
   ```

3. **Install SpikeInterface 0.95.1**
   Klustakwik requires SpikeInterface <0.96. Use:
   ```
   pip install spikeinterface[full]==0.95.1
   ```

4. **Install Spyder**
   I installed spyder via the anaconda navigator menu, this automatically set to Spyder 5.3.3.

5. **(optional?) in spyder**
   when running import spikeinterface.full as si in Spyder, I got the following warning
   ```
   C:\Users\susan\anaconda3\envs\klusta\lib\site-packages\tqdm\auto.py:21:
   TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
   https://ipywidgets.readthedocs.io/en/stable/user_install.html from .autonotebook
   import tqdm as notebook_tqdm
   ```
   It looks like this prevents spikeinterface from detecting klusta correctly. The warning seems to depend on the version of Spyder.
   Solve this by running `pip install ipywidgets` from the command prompt for the klusta environment.

Using other sorters

Most other sorters will work with Python >3.7. The latest version of SpikeInterface requires Python ≥3.8.

If you're not using klustakwik, you can just install the latest versions instead.

The code for Axona listed in this document should work with newer versions of SpikeInterface and Python, but I have not tested this extensively.

Installation (if no Klustakwik is needed)

Spyder and Anaconda navigator

I used Spyder for most procedures in this document. As someone who is more familiar with Matlab, I find Spyder the most convenient way to interface with Python.
It should be possible to run all python scripts in Jupyter notebooks as well. It is currently not possible to use my code as functions from the command prompt. I do not know how to convert them for this use yet, but it should be possible and I'll probably make this update in the future.

Installing SpikeInterface with klustakwik

Klustakwik and klusta are old legacy software require the following installations of Python and SpikeInterface.

1. **Create a conda environment with Python 3.8 or newer**
   I used the anaconda navigator for this and created the environment *SIlatest.* You can name your environment whatever you like.

2. **Install SpikeInterface and ProbeInterface**
   In the conda command prompt, activate your environment:
   ```
   conda activate klusta
   ```

   Then install spikeinterface and and probeinterface with pip. Type:
   ```
   pip install spikeinterface
   pip install probeinterface
   ```

3. Install some other useful packages
   Use pip install for the following: scipy, tqdm, numpy, numba. Some of these may already have been installed with SpikeInterface.

4. **Install Spyder**
   I installed spyder via the anaconda navigator menu.

5. **(optional?) in spyder**
   when running import spikeinterface.full as si in Spyder, I got the following warning
   ```
   C:\Users\susan\anaconda3\envs\klusta\lib\site-packages\tqdm\auto.py:21:
   TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
   https://ipywidgets.readthedocs.io/en/stable/user_install.html from .autonotebook
   import tqdm as notebook_tqdm
   ```

It looks like this prevents spikeinterface from detecting klusta correctly. The warning seems to depend on the version of Spyder.
Solve this by running `pip install ipywidgets` from the command prompt for your environment.

Preprocessing Axona .set files

Each recording from Axona consists of a large .bin file with the recorded signals, and a small .set file with information about the recording. SpikeInterface uses the .set file to determine recording parameters. The collectMask lines are used to determine the presence of tetrode signals. If collectMask is set to 1, the tetrode will be read, and if it is set to 0, the signal will not be read.

For example:

```
collectMask_1 1
collectMask_2 0
collectMask_3 1
```

In this case signals for tetrode 1 and 3 will be read, but not signals from tetrode 2.

In Jezek lab recordings, collectMask_1 is set to 1, and all others to 0. To read.bin files with all 16 tetrodes in spikeinterface, open the .set file in a text editor and change collectMask_1 to collectMask_16 from 0 to 1. If there are 8 tetrodes in a recording, only change collectMask_1 to collectMask_8.

You will need to do this for each .set file.

Extracting LFP traces from Axona .bin files

This is a procedure for extracting LFP traces from selected tetrodes for a number of .bin files. Signals from separate .bin files can be merged if desired. Signals are downsampled by a factor 4 (from 24kHz to 6kHz in our recordings), and then stored as variable `Sig` in a separate .mat file for each tetrode.

It is not possible to merge and output an entire multi-hour recording in one step with this script (the file will be too large). The best way to get traces for an entire recording is to split the recording into multiple .bin file groups and merge those.

This script does not take bad channels into account and does not skip them. All 4 channels will be extracted for the selected tetrode.

Then, in matlab, merge the downsampled recordings (optional: average the signals per tetrode first).

All 4 channels for the tetrode are extracted, there are no channels excluded in this script. The script assumes file names BaseName_01.bin and BaseName_01.set for the files.

To downsample by a different amount, edit signal.decimate() in line 61 of ReadAxonaBins.py

Use ReadAxonaBins.py (script)

Inputs:

allnames: a list with session names for each recorded to-be-merged session. Should be the same size as allsess. E.g: allnames = ["pre","post1"]

allsess: list that defines which .bin files are read and merged.
E.g: allsess = [[1,2],[4,5]] . In this case, .bin files 1 and 2 will be read and merged, and bin files 4 and 5 will be read and merged.

Tetnums: numbers of the tetrodes you want to process. These numbers are 1-based, tetrode 1 is the first tetrode (just like on the drive).
E.g: tetnums = [1, 5, 7] will export signals for tetrodes 1, 5, and 7.

InFolder: path to the directory with the .bin files you want to process. E.g: InFolder = "E:/SEPSIS/RAW/BIN/HD214-2004/"

OutFolder: path to the directory where the generated .mat files are stored. E.g: OutFolder = "E:/SEPSIS/RAW/SIconvert/"

BaseName: the common part of .bin file names for the processed recording. This name is also used as an output file name. E.g: BaseName ="HD214-2004_" for HD214-2004_01.bin, HD214-2004_02.bin, HD214-2004_03.bin etc.

Outputs:

A .mat file containing variable Sig (4 x N samples). E.g: HD214-2004_tt1_pre.mat

Extracting LFP traces from OpenEphys files

LFP traces from OpenEphys files can be extracted into Matlab with the load_open_ephys_data.m function made by OpenEphys.

```
[data, timestamps, info] = load_open_ephys_data(filename);
```

The filename input variable is the complete path to the input file (E.g. E:\MyData\ 2024-01-11_11-59-42\100_CH25.continuous for traces, or e.g. E:\MyData\ 2024-01-11_11-59-42\TTp110.0n2.spikes for thresholded data. The outputs are data (full LFP or spike waveforms), timestamps, and recording information of the loaded file.

Clustering Axona and OpenEphys data with SpikeSort3D

For OpenEphys files

Spikes recorded with Open Ephys can be exported to .ntt files than can be manually clustered in SpikeSort3D. This can be done in two different ways: from thresholded recordings and by detecting spikes from the raw traces .

From thresholded data, use the OEPhysSpikes2NTT.m function in matlab. This function reads timestamps and waveforms from OpenEphys .spikes files and converts them to a neuralynx .ntt file.

First, using ExtractPeaksOEphysSI.py, the spikes are detected from .continuous files, a 32-sample waveform for each spike is collected, and timestamps (in samples, starting at 0) and waveforms are saved in a .mat file.
Second, the generated .mat file is converted to a .ntt file with OEPhysPyMat2NTT.m.

There is currently no option to merge multiple OpenEphys .continuous files in the python script. Tetrodes with fewer than 3 good wires are not processed.

If there are only 3 good wires, the good channels will be located on the first 3 channels of the .ntt file and the bad channel will be the fourth. E.g: if tetrode wires 1, 3, and 4 were recorded, the .ntt will have spikes on channel 0 (tetrode wire 1), channel 1 (tetrode wire 3) and channel 2 (tetrode wire 4). Channel 3 will only have zeros.

There is no explicit time vector in these converted recordings and in the processing of the spikes.

Use ExtractPeaksAxonaSI.py (spyder script)

Inputs:

Chanlist: Tetlist.txt is a comma separated text file that indicates good and bad wires . It contains one line per tetrode with 5 values per line (tetrode number, value for wire 0, value for wire 1, value for wire 2, value for wire 3). There is no header. Values for good wires are set to 1, bad wires are set to 0. See example in the code chapter. Tetlist.txt should be located in InFolder.

InFolder: path where the to-be-processed .bin files are stored. E.g. InFolder = "D:/SEPSIS/"

OutFolder: path where the the generated .mat files are stored. E.g. OutFolder = "D:/SEPSIS/ntt_test/"

Tetnums: numbers of the tetrodes you want to process (1-based, same as on the rat/rec). Tetnums = [2,4,5,6,7,8,9,11]

Probepath: path where the .prb files are located. The same .prb probe layout can be used for all tetrodes (there is one for a 4-wire tetrode and one for a 3-wire tetrode).

spike_thresh: setting for spike detection, the spike detection threshold is spike_thresh * median signal standard deviation. E.g: spike_thresh = 5

spike_sign: setting for spike detection, defining the direction of the detected peaks. Can be: 'neg', 'pos', 'both'. E.g: spike_sign = 'neg' detects negative peaks only.

Outputs:
A .mat file containing variables Timestamps (1 x N) with spike times for each spike (in samples) and Spikes (32 x 4 x N) with 32-sample waveforms on each tetrode channel for each spike. 8 samples are stored before to the peak, and 24 after the peak.

Use OEPhysPyMat2NTT_v2.m (matlab function)

Function: [InFile, Spikes, Features, Timestamps, ScFac, Fs] = OEPhysPyMat2NTT_v2(InPath, InFile, OutPath, Fs, addScFac)

Requirements: This function uses the Mat2NlxSpike provided by Neuralynx

Inputs:
Inpath: path where .mat file is located. E.g: Inpath = ' M:\Leemburg\OEphysTEST'

Outpath: path where .ntt file will be generated.E.g. Outpath = ' M:\MyRec\ntt'

InFile: name of the input file. E.g: Filename = 'TT5.mat'. This is also used for the output file name.

addScFac: Option to add a scaling factor for the waveforms. This helps when displaying waveforms in SpikeSort3D. Set to 0 to keep waveforms at original amplitudes, set to 1 to add automatic scaling. If automatic scaling is used, the largest absolute amplitude value of the extracted spikes is set to 20000 and everything else is scaled accordingly.

Fs: sampling rate in Hz.

Output:
An .ntt file with the signals from the selected .spikes file. Peak and valley values are added for each spike. The other features used for clustering are calculated on the fly by SpikeSort3D (they are not included in the .ntt files generated by the Jezek lab Cheetah system either). Scaling for the data when viewed in SpikeSort3D does not match the true signal amplitudes, but is scaled according to NLX system settings (maximum value is set to 32767).
Scaled and inverted spike waveforms (Spikes), timestamps in microseconds (Timestamps), sampling rate (Fs), Features (peak and trough values for each spike), and scaling factor (ScFac) outputs are shown in the matlab workspace.

Use OEPhysSpikes2NTT_v2.m (matlab function)

Function: [InFile, ScFac,data,Features,timestamps] = OEPhysSpikes2NTT_v2 (InPath, InFile, OutPath, wv_plot, spk_plot, addScFac)

Requirements: This function uses the Mat2NlxSpike provided by Neuralynx and read_open_ephys_data provided by OpenEphys

Inputs:

Inpath: path where .spikes file is located. E.g: Inpath = 'M:\2024-01-11_11-59-42\Record Node 112'

Outpath: path where .ntt file will be generated.E.g. Outpath = ' M:\MyRec\ntt'

Filename: name of the input file. E.g: Filename = ' TTp110.0n2.spikes'. This is also used for the output file name.

wv_plot: set to 1 to plot 500 random waveforms from the loaded file. Set to 0 to skip that.

Spk_plot: set to 1 to plot wire-pair scatter plots for 500 random spikes from the loaded file. Plots show peak spike values. If wv_plot is set to 1, the spikes selected for both plots are the same. Set to 0 to skip this.

addScFac: Option to add a scaling factor for the waveforms. This helps when displaying waveforms in SpikeSort3D. Set to 0 to keep waveforms at original amplitudes, set to 1 to add automatic scaling. If automatic scaling is used, the largest absolute amplitude value of the extracted spikes is set to 20000 and everything else is scaled accordingly.

Output:

An .ntt file with the signals from the selected .spikes file. Peak and valley values are added for each spike. The other features used for clustering are calculated on the fly by SpikeSort3D (they are not included in the .ntt files generated by the Jezek lab Cheetah system). Scaling for the data when viewed in SpikeSort3D does not match the true signal amplitudes, but is scaled according to NLX system settings (maximum value is set to 32767).

Scaled and inverted spike waveforms (data), timestamps in microseconds (timestamps), sampling rate (Fs), Features (peak and trough values for each spike), and scaling factor (ScFac) outputs are shown in the matlab workspace.

For Axona files

Spikes recorded with Axona can be exported to .ntt files than can be manually clustered in SpikeSort3D. First, using ExtractPeaksAxonaSI.py, the spikes are detected from .bin files, a 32-sample waveform for each spike is collected, and timestamps (in samples, starting at 0) and waveforms are saved in a .mat file. Second, the generated .mat file is converted to a .ntt file with writeNTTfromSI_ax.m.

There is currently no option to skip bin files in the python script. Tetrodes with fewer than 3 good wires are not processed by the python script.

If there are only 3 good wires, the good channels will be located on the first 3 channels of the .ntt file and the bad channel will be the fourth. E.g: if tetrode wires 1, 3, and 4 were recorded, the .ntt will have spikes on channel 0 (tetrode wire 1), channel 1 (tetrode wire 3) and channel 2 (tetrode wire 4). Channel 3 will only have zeros.

There is no explicit time vector in these converted recordings and in the processing of the spikes. So, if two bin files are processed, the first sample of the second file follows directly after the last sample of the first file. Please keep this in mind if the clustered recording needs to be separated in time periods later (lengths of each .bin file are stored in the .set file, but starting times may not be). As far as I can tell, it is not possible to keep recording segments separated in an .ntt file, although it may be possible to generate some kind of event file.

For automated clustering, recorded segments (e.g. bin files) can be processed together and kept intact via SpikeInterface.

Use ExtractPeaksAxonaSI.py (spyder script)

Inputs:

Tetlist: Tetlist.txt is a comma separated text file that indicates good and bad wires . It contains one line per tetrode with 5 values per line (tetrode number, value for wire 0, value for wire 1, value for wire 2, value for wire 3). There is no header. Values for good wires are set to 1, bad wires are set to 0. See example in the code chapter. Tetlist.txt should be located in InFolder.

InFolder: path where the to-be-processed .bin files are stored. E.g. InFolder = "D:/SEPSIS/"

OutFolder: path where the the generated .mat files are stored. E.g. OutFolder = "D:/SEPSIS/ntt_test/"

BaseName: the common part of .bin file names for the processed recording. This name is also used as an output file name. E.g: BaseName ="HD214-2004_" for HD214-2004_01.bin, HD214-2004_02.bin, HD214-2004_03.bin etc.

numsess: the number of sessions/.bin files in the recording. E.g. numsess = 4, will read bin files 01, 02, 03 and 04.

Tetnums: numbers of the tetrodes you want to process (1-based, same as on the rat/rec). Tetnums = [2,4,5,6,7,8,9,11]

spike_thresh: setting for spike detection, the spike detection threshold is spike_thresh * median signal standard deviation. E.g: spike_thresh = 5

spike_sign: setting for spike detection, defining the direction of the detected peaks. Can be: 'neg', 'pos', 'both'. E.g: spike_sign = 'neg' detects negative peaks only.

Outputs:
A .mat file containing variables Timestamps (1 x N) with spike times for each spike (in samples) and Spikes (32 x 4 x N) with 32-sample waveforms on each tetrode channel for each spike. 8 samples are stored before to the peak, and 24 after the peak.

Use writeNTTfromSI_ax.m (matlab function)

Function: [Filename] = writeNTTfromSI_ax(Inpath, Outpath, Filename)

Requirements: This function uses the Mat2NlxSpike provided by Neuralynx

Inputs:
Inpath: path where .mat file generated by ExtractPeaksAxonaSI.py is located. E.g: Inpath = 'D:\SEPSIS\ntt_test'

Outpath: path where .ntt file will be generated.E.g. Outpath = 'D:\SEPSIS\ntt_test'

Filename: name of the input file (generated by ExtractPeaksAxonaSI.py) E.g: Filename = 'HD190-1511_tt2.mat'. This is also used for the output file name.

Output:
An .ntt file with the signals in the selected .mat file. Peak and valley values are added for each spike. I think the other features used for clustering are calculated on the fly by SpikeSort3D (they are not included in the .ntt files generated by the Jezek lab Cheetah system).

Clustering with Klustakwik and Phy

- in progress -

Code

Make a single tetrode .prb file

The following code can be used to generate a .prb file with a single tetrode. For a 3-wire tetrode, you can edit the resulting .prb file (remove one wire).

```
from probeinterface import ProbeGroup, Probe, generate_tetrode, write_prb, read_prb

Outfile = 'C:/Users/susan/Desktop/klustatest/tet4_probe.prb'


# 4-wire tetrode

probegroup = ProbeGroup()

new_chans = [0, 1, 2, 3]

tet4_probe = generate_tetrode()

tet4_probe.set_device_channel_indices(new_chans)

probegroup.add_probe(tet4_probe)

write_prb(Outfile, probegroup)
```

Tetlist.txt

This is an example of Tetlist.txt. In this file, tetrode 2 has 2 bad wires (wire 1 and 2), tetrode 4 has one bad wire (wire 3), and tetrode 12 has 3 bad wires (wire 0, wire 2 and wire 3).

Note: all 16 tetrodes must be listed in Tetlist.txt.

```
1,1,1,1,1
2,1,0,0,1
3,1,1,1,1
4,1,1,1,0
5,1,1,1,1
6,1,1,1,1
7,1,1,1,1
8,1,1,1,1
9,1,1,1,1
10,1,1,1,1
11,1,1,1,1
12,1,0,0,0
13,1,1,1,1
14,1,1,1,1
15,1,1,1,1
16,1,1,1,1
```

ReadAxonaBins.py

```
import spikeinterface as si
import spikeinterface.extractors as se
from scipy.io import savemat
import numpy as np
from scipy import signal
##
allnames = ["pre","post1","post2a", "post2b","post2c"]
allsess = [[1,2],[4,5],[7,8,9,10],[11,12,13,14],[15,16,17,18]]

Tetnums = [2,4,5,6,7,8,9,11]

InFolder = "E:/SEPSIS/RAW/BIN/HD214-2004/"
OutFolder = "E:/SEPSIS/RAW/SIconvert/"
BaseName ="HD214-2004_"

##
for sess in range(len(allsess)):

    Sessnums = allsess[sess]
    RecName = allnames[sess]

    files = list()
    #make file list
    for f in range(np.size(Sessnums)):
         InName = InFolder+BaseName+"%02d" % Sessnums[f]+".bin"
         list.append(files, InName)


    recording_list = list()
    for f in range(np.size(Sessnums)):
        #read axona file
        list.append(recording_list, se.AxonaRecordingExtractor(files[f]))

    ##
    MergeRec = si.concatenate_recordings(recording_list)

    ##
    TheseSigs = []
    for tet in range(np.size(Tetnums)):

        thistet = Tetnums[tet]
        gettet = thistet-1
        a = 4;
        chanIDs1 = gettet*a
        chanIDs1 = chanIDs1
        chanIDs2 = chanIDs1 + 4
        chanlist = range(chanIDs1,chanIDs2)
        chanlist = ["{:01d}".format(x) for x in chanlist]

        TheseSigs = MergeRec.get_traces(channel_ids = chanlist)
        TheseSigs = signal.decimate(TheseSigs, 4, axis = 0)

        thistet = str(thistet)
        outname = OutFolder+BaseName+"tt"+thistet +"_"+ RecName + ".mat"
        savemat(outname,{'Sig': TheseSigs})

        del TheseSigs
        TheseSigs = []


print('\a')
```

ExtractPeaksAxonaSI.py

```
import spikeinterface as si
import spikeinterface.extractors as se
from scipy.io import savemat
import numpy as np
from spikeinterface.sortingcomponents.peak_detection import detect_peaks
from spikeinterface.preprocessing import bandpass_filter
from tqdm import tqdm

InFolder = "D:/SEPSIS/"
OutFolder = "D:/SEPSIS/ntt_test/"
BaseName ="HD190-1511_"

numsess = 2 #number of sessions/ .bin files
Tetnums = [2,4,5,6,7,8,9,11] #tetrodes to process (1-based, same as on the rat/rec)
spike_thresh = 5 # detection threshold for spike detection is spike_thresh* signal SD
spike_sign = 'neg' #detect negative peaks only. can be: 'neg', 'pos', 'both'

# !!! need to manually edit the .set files so that all channels get read.
##############################################################################

allsess = range(numsess) #count sessions zero-based

# make file list
files = list()
for f in allsess:
   thissess = allsess[f]+1 #1-based session number
   InName = InFolder+BaseName+"%02d" % thissess+".bin"
   list.append(files, InName)

# make list of SI extractor recordings
recording_list = list()
for f in allsess:
   list.append(recording_list, se.AxonaRecordingExtractor(files[f]))

# merge recording objects
MergeRec = si.concatenate_recordings(recording_list)

# add highpass filter
MergeRec_f = bandpass_filter(MergeRec, freq_min=600.0, freq_max=6000.0, margin_ms=5.0, dtype=None)

# detect peaks per tetrode and get trace segments for waveforms
for tet in range(np.size(Tetnums)):


        thistet = Tetnums[tet]

        print('processing TT'+str(thistet))

        gettet = thistet-1
        a = 4;
        chanIDs1 = gettet*a
        chanIDs2 = chanIDs1 + 4
        chanlist = range(chanIDs1,chanIDs2)
        chanlist = ["{:01d}".format(x) for x in chanlist]

        TheseSigs = MergeRec_f.channel_slice(channel_ids = chanlist)
       #detect peaks
               detectradius = 30 #tetrode map is 10x10um square, this should capture all spikes in this
        #channelgroup
        TetPeaks = detect_peaks(thistet_f, method='locally_exclusive', pipeline_nodes=None,
        gather_mode='memory', folder=None, names=None,
```

```
        peak_sign=spike_sign, detect_threshold=spike_thresh, exclude_sweep_ms=0.1, radius_um=detectradius)
        #TetPeaks = detect_peaks(TheseSigs, method='by_channel', pipeline_nodes = None,
#gather_mode='memory', folder=None, names=None,
                                #peak_sign=spike_sign, detect_threshold= spike_thresh, exclude_sweep_ms =
#0.75) #exclude sweep is set to the same length as used by our NLX recordings
        #TetPeaks has fields: sample_index, channel_index, amplitude, segment_index


        # get waveforms. must be 32 samples long, I'm setting peak at sample 8 (same as NLX)
        prepeak = 8
        postpeak = 24

        print('extracting waveforms')

        allWFs = np.zeros([32,4,len(TetPeaks['sample_index'])], dtype = 'int16')
        for p in tqdm(range(len(TetPeaks['sample_index'])), desc="collecting waveforms"):
            sf = TetPeaks['sample_index'][p] - prepeak
            ef = TetPeaks['sample_index'][p] + postpeak

            thisWF = TheseSigs.get_traces(segment_index = None, start_frame = sf, end_frame = ef)

            #write only complete spike waveforms (might skip last spike if too short)
            if np.size(thisWF,0) == 32:
                allWFs[:,:,p] = thisWF

            del thisWF

        #save peaks to mat file (for later processing with Mat2NlxSpike to generate .ntt file)
        print('saving to mat file')
        outname = OutFolder+BaseName+"tt"+str(thistet) +'.mat'
        savemat(outname,{"Timestamps":TetPeaks['sample_index'], "Spikes": allWFs})
```

writeNTTfromSI_ax.m

```
function [Filename] = writeNTTfromSI_ax(Inpath, Outpath, Filename)

%% load SI data
%get waveforms and timestamps
load([Inpath,'\',Filename],'Timestamps','Spikes')

%% add fourth channel if missing
if min(size(Spikes)) == 3
    filler = zeros(1,length(Spikes));
    Spikes = [Spikes,filler];
    clear filler
end

%% write .ntt
Outname = strsplit(Filename,'.mat');
Outname = Outname{1};
NTTname = [Outpath,'\',Outname,'.ntt'];

% FieldSelectionFlags(1): Timestamps (1xN vector of timestamps, ascending
% order
% FieldSelectionFlags(2): Spike Channel Numbers
% FieldSelectionFlags(3): Cell Numbers (here: 0, no cells sorted yet)
% FieldSelectionFlags(4): Spike Features (8xN integer vector of features
% from cheetah: peaks for 4 channels and valley for 4 channels.
% FieldSelectionFlags(5): Samples 32x4xN integer matrix with the datapoints
% (waveform) for each spike for all 4 channels.
% FieldSelectionFlags(6): Header

AppendToFileFlag = 0; %new file will be created or old file will be overwritten
ExportMode = 1; %export all
FieldSelectionFlags = [1,1,1,1,1,0];

numspikes = numel(Timestamps);
ScNumbers = zeros(1,numspikes); %set to 0 (cheetah also does that)
CellNumbers = ScNumbers; %all cells in cluster 0

Features = nan(8,numel(Timestamps));
for s = 1:numel(Timestamps)
    Features(1:4,s) = max(Spikes(:,:,s));
    Features(5:8,s) = min(Spikes(:,:,s));
end

disp(['exporting to ', Outpath])
Mat2NlxSpike(NTTname, AppendToFileFlag, ExportMode, [], FieldSelectionFlags, Timestamps, ScNumbers,
CellNumbers, Features, Spikes)
disp(['created ',Filename,'.ntt'])

end
```

OpenEphysSpikes2NTT.m

```
function InFile = OEPhysSpikes2NTT(InPath,InFile,OutPath,wv_plot,spk_plot,Header)
%% load spike file
disp('loading spikes')
fn = [InPath,'/',InFile];
[data, timestamps, info] = load_open_ephys_data(fn);


%% plot 500 random waveforms from this tetrode (if mkplot is set to 1)
numspikes = numel(timestamps);
if wv_plot == 1
    numWV = 1000;%numspikes; %change for different number of waveforms in plot
    %WVsToPlot = 1:numspikes;
    WVsToPlot =randperm(size(timestamps,1),numWV);

    figure;
    PlotName = strsplit(InFile,'.spikes');
    PlotName = PlotName{1};
    sgtitle(PlotName);

    minY = min(min(min(data(WVsToPlot,:,:))));
    maxY = max(max(max(data(WVsToPlot,:,:))));
    for w = 1:4
        subplot(2,2,w)
        plot(1:40,squeeze(data(WVsToPlot,:,w)))
        hold on
        plot([8,8],[minY maxY],'k')
        title(['w ',num2str(w-1)])
        xlim([0 40])
        ylim([minY maxY])
    end
end

%% shorten waveforms, permute matrices
%data = data(:,1:32,:); %.ntt requires waveform of 32pts, not 40, OEPhys has: 8 pre-peak and 32 post-peak
samples
data = data(:,1:32,:);
data = permute(data,[2,3,1]);

timestamps = timestamps';
timestamps = timestamps*10^6; %convert to microseconds
%% make output filename
Outname = strsplit(InFile,'.');
NTTname = [OutPath,'\',Outname{1},'_',Outname{2},'.ntt'];

%% make inputs for .ntt file
 %BitVolts = 0.19499999284744262695; %from Continuous_Data.openephys
% FieldSelectionFlags(1): Timestamps (1xN vector of timestamps, ascending
% order
% FieldSelectionFlags(2): Spike Channel Numbers
% FieldSelectionFlags(3): Cell Numbers (here: 0, no cells sorted yet)
% FieldSelectionFlags(4): Spike Features (8xN integer vector of features
% from cheetah: peaks for 4 channels and valley for 4 channels.
% FieldSelectionFlags(5): Samples 32x4xN integer matrix with the datapoints
% (waveform) for each spike for all 4 channels.
% FieldSelectionFlags(6): Header

AppendToFileFlag = 0; %new file will be created or old file will be overwritten
ExportMode = 1; %export all
FieldSelectionFlags = [1,1,1,1,1,1];
```

```
ScNumbers = zeros(1,numspikes); %set to 0 (cheetah also does this)
CellNumbers = ScNumbers; %all cells in cluster 0

% Need integers for .ntt waveforms. Convert as: wv*10?
%data = data*100;
data = round(data); %other option
data = -data; %invert signal

%
Features = nan(8,numspikes);
for s = 1:numspikes
    Features(1:4,s) = max(data(:,:,s),[],1);
    Features(5:8,s) = min(data(:,:,s),[],1);
end

%% plot spikes scatterplot (peaks only)
if spk_plot ==1

    if wv_plot ~=1 %if the waveforms are plotted, use the same spikes for the scatterplot
        numWV = 1000; %change for different number of waveforms in plot
        WVsToPlot =randperm(size(timestamps,1),numWV);
    end

    figure;
    PlotName = strsplit(InFile,'.spikes');
    PlotName = PlotName{1};
    sgtitle(PlotName);

    maxY = max(max(max(data(:,:,WVsToPlot))));
    subplot(2,3,1)
    plot(Features(1,WVsToPlot),Features(2,WVsToPlot),'.k')
    xlabel('w 0')
    ylabel('w 1')
    xlim([0 maxY])
    ylim([0 maxY])

    subplot(2,3,2)
    plot(Features(1,WVsToPlot),Features(3,WVsToPlot),'.k')
    xlabel('w 0')
    ylabel('w 2')
    xlim([0 maxY])
    ylim([0 maxY])

    subplot(2,3,3)
    plot(Features(1,WVsToPlot),Features(4,WVsToPlot),'.k')
    xlabel('w 0')
    ylabel('w 3')
    xlim([0 maxY])
    ylim([0 maxY])

    subplot(2,3,4)
    plot(Features(2,WVsToPlot),Features(3,WVsToPlot),'.k')
    xlabel('w 1')
    ylabel('w 2')
    xlim([0 maxY])
    ylim([0 maxY])

    subplot(2,3,5)
    plot(Features(2,WVsToPlot),Features(4,WVsToPlot),'.k')
    xlabel('w 1')
    ylabel('w 3')
    xlim([0 maxY])
    ylim([0 maxY])
```

```
    subplot(2,3,6)
    plot(Features(3,WVsToPlot),Features(4,WVsToPlot),'.k')
    xlabel('w 2')
    ylabel('w 3')
    xlim([0 maxY])
    ylim([0 maxY])
end

%%
disp(['exporting to ', OutPath])
Mat2NlxSpike(NTTname, AppendToFileFlag, ExportMode, [], FieldSelectionFlags, timestamps, ScNumbers,
CellNumbers, Features, data, Header)

disp(['created ',[OutPath,'\',Outname{1},'_',Outname{2}],'.ntt'])

end
```

OEphysPyMat2NTT_v2.m

```
function [InFile,Spikes,Features,Timestamps, ScFac, Fs] = OEPhysPyMat2NTT_v2(InPath,InFile,OutPath,
Fs,addScFac)
%% load spike file
disp('loading spikes')
load([InPath,'/',InFile],'Spikes','Timestamps');

%% convert to correct formats
%Fs = 30000; %sampling rate
microsamp = (10^6)/Fs; %microsecond per sample
Timestamps = Timestamps*microsamp; %convert to microseconds
Timestamps = double(Timestamps);
numspikes = numel(Timestamps);

Spikes = double(Spikes); % convert to doubleif Spikes and Timestamps are not double, everything BSODs.
Spikes = -Spikes; %flip waveforms

%% add scaling (optional)
if addScFac == 1

    maxval = max(max(max((Spikes))));
    ScFac = 25000/maxval;
    ScFac = round(ScFac);

    Spikes = Spikes*ScFac;
else
    ScFac = 1;
end

%% make output filename
Outname = strsplit(InFile,'.');
NTTname = [OutPath,'\',Outname{1},'.ntt'];

%% make inputs for .ntt file

%BitVolts = 0.19499999284744262695; %from Continuous_Data.openephys

% FieldSelectionFlags(1): Timestamps (1xN vector of timestamps, ascending
% order
% FieldSelectionFlags(2): Spike Channel Numbers
% FieldSelectionFlags(3): Cell Numbers (here: 0, no cells sorted yet)
% FieldSelectionFlags(4): Spike Features (8xN integer vector of features
% from cheetah: peaks for 4 channels and valley for 4 channels.
% FieldSelectionFlags(5): Samples 32x4xN integer matrix with the datapoints
% (waveform) for each spike for all 4 channels.
% FieldSelectionFlags(6): Header

AppendToFileFlag = 0; %new file will be created or old file will be overwritten
ExportMode = 1; %export all
FieldSelectionFlags = [1,1,1,1,1,0];

ScNumbers = zeros(1,numspikes); %set to 0 (cheetah also does this)
CellNumbers = ScNumbers; %all cells in cluster 0

Features = nan(8,numspikes);
for s = 1:numspikes
    Features(1:4,s) = max(Spikes(:,:,s),[],1);
    Features(5:8,s) = min(Spikes(:,:,s),[],1);
end

%%
```

```
disp(['exporting to ', OutPath])
Mat2NlxSpike(NTTname, AppendToFileFlag, ExportMode, [], FieldSelectionFlags, Timestamps, ScNumbers,
CellNumbers, Features, Spikes)

disp(['created ',[OutPath,'\',Outname{1}],'.ntt'])

end
```

ExtractSpikesOEphys.py

```
# -*- coding: utf-8 -*-
import spikeinterface.extractors as se
import numpy as np
from probeinterface import read_prb
from spikeinterface.sortingcomponents.peak_detection import detect_peaks
from spikeinterface.preprocessing import bandpass_filter
from scipy.io import savemat
from tqdm import tqdm


InFolder = "E:/Test1/2024-01-29_12-16-36/Record Node 107"
OutFolder = "C:/Users/roychoun/Desktop/klustatest/"
Probepath ="C:/Users/roychoun/Desktop"
ChanList = 'KKtetlist2.txt' # text file listing good and bad channels
TetList = [1,2,3] #analyse only these tetrodes (1-based, as on drive)

spike_thresh = 5 # detection threshold for spike detection is spike_thresh* signal SD
spike_sign = 'pos' #detect peaks. can be: 'neg', 'pos', 'both'

#############################################################################################
# assign channel numbers, group by tetrode
tetgrouping = np.array([2,2,2,2,3,3,3,3,7,7,7,7,6,6,6,6,5,5,5,5,4,4,4,4,0,0,0,0,1,1,1,1])

# read bad channel list, set bad channels to 17
tetlist = np.loadtxt(InFolder + '/' + ChanList,
                 delimiter=",")

for tetnum in range(8):
        thistet = np.where(tetgrouping==tetnum)
        thistet = np.array(thistet)
        thesewires = tetlist[tetnum,1:5]
        badwires = np.where(thesewires==0)
        badwires = np.array(badwires)
        badchans = thistet[0][badwires]
        tetgrouping[badchans]=17

# read recfile, add grouping labels
MyRec = se.OpenEphysLegacyRecordingExtractor (InFolder)
MyRec.set_channel_groups(channel_ids = MyRec.get_channel_ids(),groups = tetgrouping)
all_chan_ids = MyRec.get_channel_ids()

# select a tetrode
for tetnum in TetList:

    tet_chan_ids = all_chan_ids[np.where(tetgrouping == tetnum-1)]
    tetname = TetList[tetnum-1]

    if np.size(tet_chan_ids)>2:

        # 4-wire tetrode
        if np.size(tet_chan_ids) == 4:
           new_chans = [0,1,2,3]
           probename = "tet4_probe.prb"

        # 3-wire tetrode
        if np.size(tet_chan_ids) == 3:
            new_chans = [0,1,2]
            probename = "tet3_probe.prb"

    myprobe = read_prb(Probepath + "/" + probename)
```

```
    #select channels and add probe
    thistet = MyRec.channel_slice(tet_chan_ids, renamed_channel_ids=new_chans)
    thistet = thistet.set_probegroup(myprobe)

    # preprocess (filter)
    thistet_f = bandpass_filter(thistet, freq_min=600, freq_max=6000)

    #detect peaks
    detectradius = 30 #tetrode map is 10x10um square, thsi should capture all spikes in this channelgroup
    TetPeaks = detect_peaks(thistet_f, method='locally_exclusive', peak_sign=spike_sign,
detect_threshold=spike_thresh, exclude_sweep_ms=0.1, local_radius_um=detectradius, noise_levels=None,)

    # get waveforms. must be 32 samples long, I'm setting peak at sample 8 (same as NLX)
    prepeak = 8
    postpeak = 24

    print('extracting waveforms')

    allWFs = np.zeros([32, 4, len(TetPeaks['sample_ind'])], dtype='int16')
    for p in tqdm(range(len(TetPeaks['sample_ind'])), desc="collecting waveforms"):
        sf = TetPeaks['sample_ind'][p] - prepeak
        ef = TetPeaks['sample_ind'][p] + postpeak

        thisWF = thistet_f.get_traces(segment_index=None, start_frame=sf, end_frame=ef)

        #write only complete spike waveforms (might skip last spike if too short)
        if np.size(thisWF, 0) == 32:
           allWFs[:, :, p] = thisWF

        del thisWF

   #save peaks to mat file (for later processing with Mat2NlxSpike to generate .ntt file)
    print('saving to mat file')
    outname = OutFolder+"tt"+str(tetname) + '.mat'
    savemat(outname, {"Timestamps":TetPeaks['sample_ind'], "Spikes": allWFs})
```