spike2py Documentation

Release 0.1

Martin Heroux

Mar 23, 2019

CONTENTS

Python module that faciliates working with Spike2 data files exported to Matlab format.

The project is hosted on GitHub here.

CHAPTER ONE

OVERVIEW

Spike2 is the software used by many scientists to collect data using a Cambridge Electronics Design (CED) data aquisition boards. While some scientists use Spike2 to analyse their data, other scientists prefer to export their data to other programs such as Matlab or Python.

There are ways to directly open Spike2 data files in Python. For example, Neo is a package for analysing electrophysiology data that supports reading a wide range of neurophysiology file formats, including Spike2 files. However, package updates have been known to cause issues with reading Spike2 files. Moreover, the Neo framework was specifically designed to handle data from electrophysiological experiments (cellular and animal recordings; it is less well suited, or simply overkill, for data collected for life sciences, biomechanics, and human neurophysiology experiments.

As an alternative, Spike2 data can be exported to Matlab data files (.mat). While this requires the user to export data prior to opening it in Python, this approach has some advantages. First, CED makes available a Spike2 script that can batch export all Spike2 data files contained within a folder. A copy of this script file is available here. Second, opening Matlab data files has been supported for a long time in Python through the scipy.io module.

The one downside to this approach is that when these Matlab data files are opened in Python, it is not intuitive how the data are organized, nor where the various details of your signal might be stored (e.g. sampling rate, sample times, etc). Rarely would someone ever work directly with the default structure of nested numpy arrays in a dictionary.

The present module provides a simple interface to signals and triggers recorded in Spike2. The main building blocks are a `Signal` class and a `Trial` class. The `Trial` class can store all the signals recorded during a trial, as well as other details related to that trial. Basic signal processessing, such as calibration, filtering, offset removal, can be applied to the signals. There is also a default routine for surface electromyography (EMG) recordings (filtering, rectifying, envelop filtering). The module include several helper classes to streamline how `Trial` and `Signal` are accessed.

CHAPTER

TWO

TUTORIAL

2.1 Basic Example

We will demonstate the basic usage of *spike2py* using data from a simple experiment. The data was collected in Spike2 and then exported as a Matlab data file using Batch_export_MATLAB.s2s. The trial lasted approximately 10s and included three signals: a trigger signal, an EMG signal, and a pressure signal.

The trigger signal was recorded as a *trigger*. This means Spike2 saved the times when the trigger pulse occurred. The EMG and pressure signal were both recorded as *waveform* signals, which is what Spike2 calls continues signals sampled at a given sampling rate.

Here is the basic structure to retrieve the data and obtain a *Trial* object. *SigInfo* will be used in its most basic form to give *spike2py* the required information to properly import our signals. We will also use *Trial* to contain all the data and information about our trial, including the signals themselves.

```python
import spike2py

exp_cond = 'maximum pressure trial 1'
filename = 'max_push1.mat'

trig_info = spike2py.SigInfo(name='trig',
                             stype='trig',
                             s2name='trig',
                             )

emg_info = spike2py.SigInfo(name='biceps',
                            stype='sEMG',
                            s2name='EMG',
                            )

pressure_info = spike2py.SigInfo(name='pressure',
                                 stype='waveform',
                                 s2name='signal',
                                 )

signals = [trig_info, emg_info, pressure_info]

trial_info = spike2py.TrialInfo(cond=exp_cond,
                                path='./tutorials/',
                                filename=filename,
                                signals=signals,
                                )

trial = spike2py.Trial(trial_info)
```

We now have a *trial* object that contains information about our trial, and the various signals we recorded.

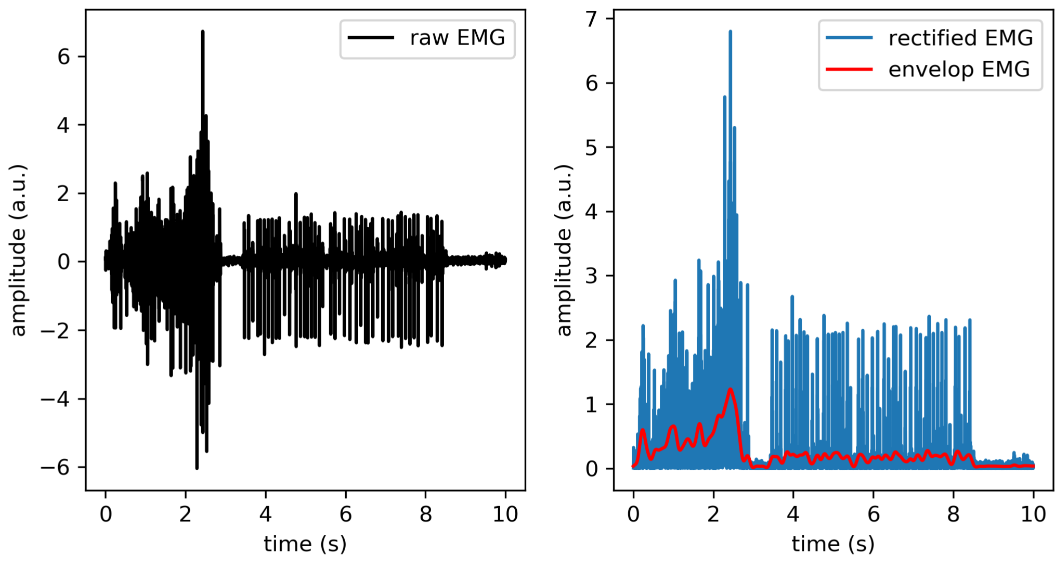
```
>>> trial.cond
'maximum pressure trial 1'
>>> trial.filename
'max_push1.mat'
>>> trial.sig
{'trig': Signal(sig=sig, name='trig_info', s2name='trig', stype='trig'),
 'biceps': Signal(sig=sig, name='biceps', s2name='EMG', stype='sEMG'),
 'pressure': Signal(sig=sig, name='pressure', s2name='signal', stype='waveform')}
```

We can access our data to process or plot. For example, we can inspect the trigger times.

```
>>> trial.sig['trig'].times
array([[0.701],
       [3.501],
       [3.801],
       [4.701],
       [4.901],
       [5.101],
       [5.301],
       [5.501],
       [5.701],
       [5.901],
       [6.101],
       [6.301],
       [6.501]])
```

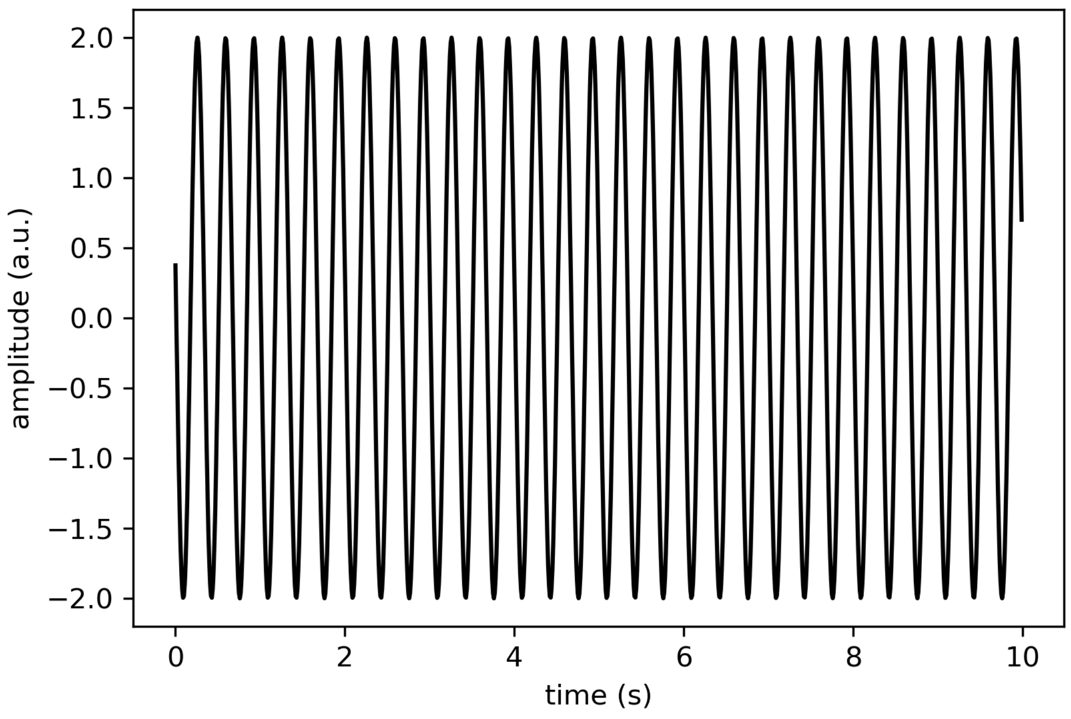
We can plot the biceps EMG data. Note that by specifying *stype=sEMG* for our EMG data, `spike2py` automatically creates a rectified and envelop version of our EMG data.:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4), dpi=300)
plt.subplot(121)
plt.plot(trial.sig['biceps'].times, trial.sig['biceps'].raw, 'k', label='raw EMG')
plt.ylabel('amplitude (a.u.)')
plt.xlabel('time (s)')
plt.legend()
plt.subplot(122)
plt.plot(trial.sig['biceps'].times, trial.sig['biceps'].rect, label='rectified EMG')
plt.plot(trial.sig['biceps'].times, trial.sig['biceps'].envel, 'r', label='envelop EMG
↪')
plt.ylabel('amplitude (a.u.)')
plt.xlabel('time (s)')
plt.legend()
```



We can also plot the pressure.

```
>>> plt.figure(figsize=(6, 4), dpi=300)
>>> plt.plot(trial.sig['pressure'].times, trial.sig['pressure'].raw, 'k')
>>> plt.ylabel('amplitude (a.u.)')
>>> plt.xlabel('time (s)')
```



2.2 Intermediate example

We will re-use the data from the previous example and learn more about *SigInfo*, understand how directly create *Signal* objects (when exploring new data and debugging your code), and better understand how *Trial* simplifies things when many signals were recorded.

2.2.1 Signals

Signal Information

We will create *SigInfo* ojbects to provide information about each of the signals that were recorded. As described in the advanced example, we can include information about how to filter or calibrate our signal, as well as remove an offset. But for this first example we will include the minimum information required.:

```
import spike2py as spk2
trig_info = spk2.SigInfo(name='trig_info', stype='trig', s2name='trig')
emg_info = spk2.SigInfo(name='biceps', stype='sEMG', s2name='EMG')
pressure_info = spk2.SigInfo(name='pressure', stype='waveform', s2name='signal')
```

For each signal we include:

- **name**: Informative name.
- **stype**: Signal type for spike2py (different to the signal types from Spike2). Tells *spike2py* how to import the signals and whether to perform some default signal processing (i.e. filtering, rectification and envolop for *sEMG*),
- **s2name**: Name of signal used when recording from Spike2.

Let's inspect the information related to our three signals:

```
>>> trig_info
SigInfo(name='trigger'
    stype='trig'
    s2name='trig'
    filt_cutoff=None,
    filt_order=None,
    filt_type=None,
    calib_slope=None,
    calib_offset=None,
    offset_type=None,
    offset_val=None)
>>> emg_info
SigInfo(name='biceps',
    stype='sEMG',
    s2name='EMG',
    filt_cutoff=[20, 450],
    filt_order=4,
    filt_type='bandpass',
    calib_slope=None,
    calib_offset=None,
    offset_type=None,
    offset_val=None)
>>> pressure_info
SigInfo(name='pressure',
    stype='waveform',
```

(continues on next page)

(continued from previous page)

```
    s2name='signal',
    filt_cutoff=None,
    filt_order=None,
    filt_type=None,
    calib_slope=None,
    calib_offset=None,
    offset_type=None,
    offset_val=None)
```

We can see that `SigInfo` has several additional parameters that we have not included; these are set to *None* by default. We can also see that because we set *emg* to have *stype=sEMG*, `spike2py` automatically created a 4th order butterworth band-pass filter of 20-450Hz. These values can be overridden if required.

Signal Object

The easiest way to import signals from data saved as at MATLAB file is to pass one or more `SigInfo` objects to `Trial`. The `Trial` Class handles the details of opening the MATLAB file and parsing out the data.

However, when starting out with `spike2py` or when debugging a problem, it can be useful to know how to directly obtain a `Signal`. Below is a simple example of how this can be done:

```
import os
import scipy.io as sio
import spike2py as spk2
```

We will use the *scipy.io* module to read our data from the MATLAB file:

```
filename = 'max_push1.mat'
file = os.path.join('.', 'tutorials', filename)
data = sio.loadmat(file)
```

All the data from our trial in in *data*, a dictionary object of signals and other information generated by *Spike2* about our trial and signals. For our example, we will create a `Signal` object for our trigger signal. To do so we will again create a `SigInfo`, and then pass the information it contains to `Signal`:

```
trig_info = spk2.SigInfo(name='trig', stype='trig', s2name='trig')
trig = spk2.Signal(sig=data[trig_info.s2name],
                   stype=trig_info.stype,
                   name=trig_info.name,
                   s2name=trig_info.s2name,
                   fs=trig_info.fs,
                   filtInfo=trig_info.filtInfo,
                   calibInfo=trig_info.calibInfo,
                   offsetInfo=trig_info.offsetInfo,
                   )
```

```
>>> trig
Signal(sig=sig, name='trig', s2name='trig', stype='trig')
>>> trig.name
'trig'
>>> trig.s2name
'trig'
>>> trig.stype
'trig'
>>> trig.times
```

(continues on next page)

(continued from previous page)

```
array([[0.701],
       [3.501],
       [3.801],
       [4.701],
       [4.901],
       [5.101],
       [5.301],
       [5.501],
       [5.701],
       [5.901],
       [6.101],
       [6.301],
       [6.501]])
```

Note: *trig* signals have their data –the time of the triggers– in the *.times* attribute.

Now let's repeat the process for our pressure signal:

```
pressure_info = spk2.SigInfo(name='pressure', stype='waveform', s2name='signal')
pressure = spk2.Signal(sig=data[pressure_info.s2name],
                       stype=pressure_info.stype,
                       name=pressure_info.name,
                       s2name=pressure_info.s2name,
                       fs=pressure_info.fs,
                       filtInfo=pressure_info.filtInfo,
                       calibInfo=pressure_info.calibInfo,
                       offsetInfo=pressure_info.offsetInfo,
                       )
```

```
>>> pressure
Signal(sig=sig, name='pressure', s2name='signal', stype='waveform')
>>> pressure.fs
100
>>> pressure.times[0:10]
array([0.  , 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09])
>>> pressure.raw[0:10]
array([ 3.75823975e-01,  7.62939453e-04, -3.74450684e-01, -7.35626221e-01,
       -1.07086182e+00, -1.36871338e+00, -1.61758423e+00, -1.80999756e+00,
       -1.93664551e+00, -1.99569702e+00])
```

Note:

- Although we did not provide the sampling frequency (*fs*) of our pressure signal, `spike2py` retreived it from the information contained in information included in the the MATLAB file.
- *waveform* signal types have their sample times in the *.times* attribute and their raw signal in the *.raw* attribute.

We will repeat the process one last time with our *emg* signal:

```
emg_info = spk2.SigInfo(name='biceps', stype='sEMG', s2name='EMG')
biceps = spk2.Signal(sig=data[emg_info.s2name],
                     stype=emg_info.stype,
                     name=emg_info.name,
```

(continues on next page)

(continued from previous page)

```
                          s2name=emg_info.s2name,
                          fs=emg_info.fs,
                          filtInfo=emg_info.filtInfo,
                          calibInfo=emg_info.calibInfo,
                          offsetInfo=emg_info.offsetInfo,
                          )
```

```
>>> biceps
Signal(sig=sig, name='biceps', s2name='EMG', stype='sEMG')
>>> biceps.fs
2000
>>> biceps.filtInfo
FiltInfo(cutoff=[20, 450], order=4, type='bandpass')
>>> biceps.times[0:10]
array([0.    , 0.0005, 0.001 , 0.0015, 0.002 , 0.0025, 0.003 , 0.0035,
       0.004 , 0.0045])
>>> biceps.raw[0:10]
array([ 0.07934562,  0.10477691,  0.14546697,  0.10782866,  0.00050863,
       -0.13885483, -0.18615703, -0.1734414 , -0.18564841, -0.23193336])
>>> biceps.proc
array([-0.00761917,  0.03972988,  0.05797267,  0.01434539, -0.09089459,
       -0.20199386, -0.25804102, -0.26125151, -0.26601012, -0.30022137])
>>> biceps.rect[0:10]
array([0.07934562, 0.10477691, 0.14546697, 0.10782866, 0.00050863,
       0.13885483, 0.18615703, 0.1734414 , 0.18564841, 0.23193336])
>>> biceps.envel[0:10]
array([0.10978258, 0.10942789, 0.10907111, 0.10871227, 0.10835142,
       0.10798861, 0.10762387, 0.10725725, 0.10688879, 0.10651856])
```

Note:

- Although we did not provide the sampling frequency (*fs*) of our biceps emg signal, `spike2py` retreived it from the information contained in information included in the the MATLAB file.
- Although we did not specify filter setting, the default is to apply a 4th order 20-450Hz bandpass Butterworth filter on the raw data.
- *sEMG* signal types have their sample times in the *.times* attribute and their raw signal in the *.raw* attribute.
- *proc* contains the EMG signal after the mean has been removed and the default (or user specified) filter has been applied.
- *rect* contains the rectified version of the *proc* signal.
- *envel* contains the envelop of the rectified signal. The default settings use a 4th order 5Hz low-pass Butterworth filter, but the `envelop` method can be re-run with a user-specified low-pass cut-off.

2.2.2 Trials

As mentioned previously, it can get quite tiresome to write code to generate individual `Signal` objects for each of the signals your recorded. Thankfully, we can use `Trial` to do all the heavy lifting for us, all we have to do is give it the right information.

Trial Information

The first thing we need to do is create a `TrialInfo` objectrig_info. The code below is a continuation of our example, which means we already created *trig_info*, *pressure_info* and *emg_info*.

```
>>> signals = [trig_info, pressure_info, emg_info]
>>> exp_cond = 'maximum pressure trial 1'
>>> path ='tutorials/'
>>> filename = 'max_push1.mat'
>>> trial_info = spike2py.TrialInfo(cond=exp_cond,
                                    path=path,
                                    filename=filename,
                                    signals=signals,
                                    )
```

`TrialInfo` requires:

- **cond**: Informative name for trial.
- **path**: Path to where the Matlab data file is stored.
- **filename**: Filename of Matlab data file, with *.mat* extension.
- **signals**: List of `SigInfo` objects.

Trial Object

Now that we have `TrialInfo` describing our current trial, lets pass it to `Trial`

```
>>> trial = spike2py.Trial(trial_info)
trial
Trial(path='tutorials/', filename='max_push1.mat', cond='maximum pressure trial 1',␣
↪sig=sig)
# signals in trial:
#        trig
#        biceps
#        pressure
>>> trial.infoSignals
[SigInfo(name='trig',
                stype='trig',
                s2name='trig',
                filt_cutoff=None,
                filt_order=None,
                filt_type=None,
                calib_slope=None,
                calib_offset=None,
                offset_type=None,
                offset_val=None),
SigInfo(name='biceps',
                stype='sEMG',
                s2name='EMG',
                filt_cutoff=[20, 450],
                filt_order=4,
                filt_type='bandpass',
                calib_slope=None,
                calib_offset=None,
                offset_type=None,
                offset_val=None),
```

(continues on next page)

(continued from previous page)

```
SigInfo(name='pressure',
                stype='waveform',
                s2name='signal',
                filt_cutoff=None,
                filt_order=None,
                filt_type=None,
                calib_slope=None,
                calib_offset=None,
                offset_type=None,
                offset_val=None)]
```

Similar to the basic tutorial, we can access our `Signal` by calling *trial.sig[sig_name]* and the various attributes we saw in the above section discussing `Signal`.

```
>>> trial.sig['trig']
Signal(sig=sig, name='trig', s2name='trig', stype='trig')
>>> trial.sig['pressure']
Signal(sig=sig, name='pressure', s2name='signal', stype='waveform')
>>> trial.sig['biceps']
Signal(sig=sig, name='biceps', s2name='EMG', stype='sEMG')
>>> trial.sig['trig'].times[0:10]
array([[0.701],
       [3.501],
       [3.801],
       [4.701],
       [4.901],
       [5.101],
       [5.301],
       [5.501],
       [5.701],
       [5.901]])
>> pressure_times = trial.sig['pressure'].times[0:10]
>> pressure_raw = trial.sig['pressure'].raw[0:10])
>>> [print(f'{t}, {p:4.2f}') for t, p in zip(pressure_times, pressure_raw]
 0.00,  0.38
 0.01,  0.00
 0.02, -0.37
 0.03, -0.74
 0.04, -1.07
 0.05, -1.37
 0.06, -1.62
 0.07, -1.81
 0.08, -1.94
 0.09, -2.00
```

CHAPTER

THREE

API REFERENCE

3.1 Signal

class `spike2py.`**`Signal`**(*sig*, *stype*, *name*, *fs=None*, *s2name=None*, *filtInfo=None*, *calibInfo=None*, *offsetInfo=None*)

Signal object

Signal and accompanying information. Most often this is a signal recorded using Spike2, but can also be an external signal that needs to be processed alongside the signal recorded with Spike2.

Parameters

- **`sig`** (`np.array or list or list of lists`) – When used in the context of the *Trial* class and a signal recorded in Spike2, *sig* will be the signal obtained from the imported Matlab file. The signal is obtained from *data[s2name]*, where *s2name* is the name of the Spike2 channel. If using with a signal not recorded in Spike2, *sig* will be either the signal on its own, or the signal and an accompanying time axis. *sig* = [*signal*, *times*] where *times* is optional.
- **`stype`** (`str`) – Signal type. *external*: signal not recorded by Spike2 but needed for analysis. *waveform* : signal sampled at regular intervals. *trig* : trigger signal *sEMG* : surface EMG signal
- **`s2name`** (`str`) – Spike2 channel name
- **`name`** (`str`) – Informative name for signal

Other Parameters

- **fs** (*int, optional*) – Sampling frequency of signal. If *times* is not present, *fs* is used to create a time axis for the signal.
- **filtInfo** (*FiltInfo, optional*) – Contains filter information
- **calibInfo** (*CalibInfo, optional*) – Contains calibration information
- **offsetInfo** (*OffsetInfo, optional*) – Contain offset information

`calibrate`(*slope=None*, *offset=None*)

Calibrate signal.

Done when signal first created if *calibInfo* parameter provided. However, it can also be performed after the signal has been created if the *slope* and *offset* parameters are provided. *calibInfo* will be added to sig.

See *CalibInfo* for details on *slope* and *offset*.

`envelop`(*cutoff=5*)

Create a signal envelop for sEMG.

Parameters **cutoff** (*int, default 5*) – Cutoff frequency for lowpass filter

filter (*cutoff=None*, *order=None*, *type=None*)

Filter signal.

Done when signal first created if *filtInfo* parameter provided. However, it can also be performed after the signal has been created if the *cutoff*, *order* and *type* parameters are provided. *filtInfo* will be added to sig.

See *FiltInfo* for details on *cutoff*, *order* and *type*.

interpolate (*name*, *sig_id*, *subset=None*, *new_x=None*, *new_fs=None*, *old_x=None*)

Linear interpolation of signal.

Parameters

- **name** (*str*) – name to give to new interpolated signal
- **sig_id** (*str*) – Select which values to interpolation (e.g. 'raw' or 'proc').
- **subset** (*list of two int and one bool, optional*) – Index values to subset signal and True/False of whether or not to zero new x-axis
- **new_x** (*np.array (optional)*) – If provided, directly used as the new_x to interp values
- **new_fs** (*int (optional)*) – If provided, new_x generated based on new sampling fq
- **old_x** (*np.array (optional, default is self.times)*) – If provided, used as the old_x

Examples

```
>>> import spike2py
```

Interpolate to a new sampling rate

```
>>> trial.sig['torque'].interpolate(name='fs15', sig_id='proc', new_fs=15)
```

Interpolate to a new x-axis

```
>>> new_x = list(range(len(trial.sig['torque'].raw_vals)))
>>> trial.sig['torque'].interpolate(name=interpolate, sig_id='raw', new_x=new_
↪x)
```

Notes

Can provide *new_x* to interpolate values to this new x-axis or times, or *fs* to interpolate to a new x-axis created with a sampling rate of *fs* Hz.

The default x-axis for the original (pre-interpolation) signal is *self.times*; however, a different *old_x* can be specified.

rem_offset (*type=None*, *val=None*)

Remove offset from signal.

Done when signal first created if *offsetInfo* parameter provided. However, it can be performed after the signal has been created if the *type* and *val* parameters are provided; *offsetInfo* will be added to signal.

See *OffsetInfo* for details on *type* and *val*.

3.2 Trial

class spike2py.**Trial**(*trialInfo*)
Class for trial.

Parameters **trialInfo** (TrialInfo) – Class containing details about trial and its signals

Examples

```
>>> import spike2py
```

```
>>> trial_pc1 = spike2py.Trial(trial_info1)  # Assumes `trial_info1` available
```

```
>>> trial_pc2 = spike2py.Trial(trial_info2)  # Assumes `trial_info2 available
>>> trial_pc1.merge('pc2', trial_pc2)
```

```
>>> us = spike2py.SigInfo(name='us', stype='external', fs=15)   # ultrasound
>>> us_sig = spike2py.Signal(us_dat, stype=us.stype, name=us.name, fs=us.fs)
>>> trial_pc1.add_sig('us', us_sig)
```

add_sig(*sig_name*, *sig*)
Add signal.

Parameters

- **sig_name** (*str*) – Short informative name for signal; will be key to retrieve signal from *Trial.sig[key]*
- **sig** (*Signal*, list, np.array, int, float, bool) – Data associated with added signal

Notes

Designed to have *sig* be a *Signal* instance, but this implementation provides flexibility to the user to add any type of signal or value.

merge(*trial*, *prefix=None*)
Merge trials.

Adds all signals from another *Trial* instance.

Parameters

- **trial** (Trial) – Trial containing signals to be merged
- **prefix** (*str, optional*) – Prefix to add to merged signal names. Default is *merged_*

3.3 SigInfo

A class to provide information about a signal recorded in Spike2. Also allows for certain pre-processing steps to be performed, such as filtering, calibrating and offset removal.

class spike2py.**SigInfo**(*name*, *stype*, *s2name=None*, *fs=None*, *filt_cutoff=None*, *filt_order=None*, *filt_type=None*, *calib_slope=None*, *calib_offset=None*, *offset_type=None*, *offset_val=None*)
Information to process signal recorded with Spike2.

Passed to spike2.TrialInfo.sig, which is a list of all sig, or a single signal.

Parameters

- **name** (*str*) – User defined name, because Spike2 channel names are not always clear.
- **stype** (*str*) – Signal type. *external'for a signal not recorded by Spike2 but needed for analysis. 'waveform* for a signal sampled at regular intervals. *trig* for a trigger signal. *sEMG* for a surface EMG signal.
- **s2name** (*str*) – Spike2 channel name.

Other Parameters

- **fs** (*int, optional*) – Sampling frequency of signal; used when *stype* is *external*
- **filt_cutoff** (*int, optional*) – Value of filter Either single value (e.g. 30) or list of 2 values (e.g. [20, 450])
- **filt_order** (*int, optional*) – Order of filter
- **filt_type** (*str, optional*) – Define type of filter: *lowpass*, *highpass*, *bandpass*, *bandstop*
- **calib_slope** (*float or int, optional*) – Slope of the calibration equation
- **calib_offset** (*float or int, optional*) – Offset of the calibration equation (i.e. intercept)
- **offset_type** (*str, optional*) – *val* to remove value provided. *mean* to remove mean of signal. *start* to remove mean of n points from start of trial.
- **offset_val** (*int, optional*) – Value required for *val* and *start* offset removal types.

Examples

```
>>> import spike2py
```

```
>>> gas = spike2py.SigInfo(name='gas', stype='sEMG', s2name='GAS',
            filt_cutoff=5, filt_order=4, filt_type='lowpass',
            calib_slope=0.5, calib_offset=55,
            offset_type='val', offset_val=22)
>>> gas
SigInfo(name='gas'
            stype='sEMG'
            s2name='GAS'
            filt_cutoff=[20, 450]
            filt_order=4
            filt_type='bandpass'
            calib_slope=0.5
            calib_offset=55
            offset_type='val'
            offset_val=22
```

3.4 TrialInfo

class spike2py.**TrialInfo**(*cond*, *path*, *filename*, *signals*)

Information to process trial recorded with Spike2.

Passed to spike2.Trial(). Assumes the data has been exported using the Spike2 script *Batch export MATLAB.2s2* available from CED.

Parameters

- **cond** (*str*) – Experimental condition (e.g. 'cs-100mvc' or 'grasp_all_digits')
- **path** (*str*) – path to .mat file exported from Spike2
- **filename** (*str*) – Full name of .mat file (e.g. 'cs-100mvc.mat')
- **signals** (*list of SignalInfo*) – Each item in list is a *SigInfo* instance. These are the sig to be analysed.

Examples

```
>>> import spike2py
```

```
>>> cond = 'grasp 10s'
>>> path = '/home/martin/Documents/grasp_study/data/'
>>> filename = 'grasp_10s.mat'
>>> signal = spike2py.SigInfo(name='FDI EMG', stype='sEMG', s2name='FDI_emg')
>>> trial_info = spike2py.TrialInfo(cond, path, filename, signal)
>>> trial_info
spike2.TrialInfo(cond='grasp 10s',
            path='/home/martin/Documents/grasp_study/data/',
            filename='grasp_10s.mat',
            signal=signal)
            # signal is an instance of spike2.SigInfo or a list of instances.
```

3.5 FiltInfo

class spike2py.**FiltInfo**(*stype=None*, *cutoff=None*, *order=None*, *type=None*)

Filter information.

Can be used independently, but most often passed to *SigInfo*.

Parameters

- **stype** (*str, optional*) – Signal type. *external*, *waveform*, *sEMG*
- **cutoff** (*int, float, or list, optional*) – Cutoff frequency of filter. Can be a single value for *lowpass* and *highpass* filters (e.g. 5), or a pair of *int* or *float* in a list (e.g. [20, 450]).
- **order** (*int, optional*) – Filter order; usually 2, 4 or 8.
- **type** (*str, optional*) – *lowpass*, *highpass*, *bandpass*, *bandstop*

Examples

```
>>> import spike2py
```

```
>>> filter = spike2py.FiltInfo(stype='sEMG')
>>> filter
FiltInfo(cutoff=[20, 450], order=4, type='bandpass')
```

```
>>> filter = spike2py.FiltInfo(cutoff=50, order=4, type='highpass')
>>> filter
FiltInfo(cutoff=50, order=4, type='highpass')
```

```
>>> filter = spike2py.FiltInfo(cutoff=[2, 10], order=8, type='bandstop')
>>> filter
FiltInfo('cutoff=[2, 10], order=8, type='bandstop')
```

Notes

The default for *stype* sEMG assumes the filter will be used on a raw signal. When a sEMG signal is created, a rectified and 5Hz lowpass envelop version is also created. If an envelop with a different cutoff frequency is desired, run *Signal.envelop(cutoff=val)* with the desired cutoff value specified.

3.6 CalibInfo

class spike2py.**CalibInfo**(*slope=None*, *offset=None*)

Calibration information.

Can be used independently, but most often passed to *SigInfo*.

Parameters

- **slope** (*int or float, optional*) – Slope of calibration equation
- **offset** (*int or float, optional*) – Offset of calibration equation (i.e. intercept)

Examples

```
>>> import spike2py
```

```
>>> calibrate = spike2py.CalibInfo(slope=54.8, offset=2.6)
>>> calibrate
CalibInfo(slope=54.8, offset=2.6)
```

3.7 OffsetInfo

class spike2py.**OffsetInfo**(*type=None*, *val=None*)

Offset removal information.

Can be used independently, but most often passed to *SigInfo*.

Parameters

- **type** (*str, optional*) – *val* to remove value provided. *mean* to remove mean of signal. *start* to remove mean of n points from start of trial.
- **val** (*int, optional*) – Value required for *val* and *start* offset removal types.

Examples

```
>>> import spike2py
```

```
>>> offset = spike2py.OffsetInfo(type='val', val=44.5)
>>> offset
OffsetInfo(type='val', val=44.5)
```

```
>>> offset = spike2py.OffsetInfo(type='mean', val='')
>>> offset
OffsetInfo(type='val', val='')
```

```
>>> offset = spike2py.OffsetInfo(type='mean', val='')
>>> offset
OffsetInfo(type='start', val=100)
```

PYTHON MODULE INDEX