CalSciPy

Release 0.3.0

Darik A. O'Neil

2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

CalSciPy contains a variety of useful methods for handling, processing, and visualizing calcium imaging data. It's intended to be a collection of useful, well-documented functions often used in boilerplate code alongside software packages such as Caiman, SIMA, and Suite2P.

1.1 Motivation

I noticed I was often re-writing or copy/pasting a lot of code between environments when working with calcium imaging data. I started this package ~so I don't have to~ so you don't have to. No more wasting time writing 6 lines to simply preview your tiff stack, extract a particular channel, or bin some spikes. No more vague exceptions or incomplete documentation when re-using a hastily-made function from 2 months ago. Alongside these time-savers, I've also included some more non-trivial methods that are particularly useful.

1.2 Limitations

The current distribution for the package is incomplete and partially tested

CHAPTER

TWO

INSTALLATION

2.1 Full Install

Enter `pip install CalSciPy` in your terminal.

2.2 Partial Install

Enter `pip install CalSciPy-<subpackage>` in your terminal.

CHAPTER

THREE

OVERVIEW

- *Bruker*
- *Coloring*
- *Event Processing*
- *Input/Output Tools (I/O)*
- *Image Processing*
- *Interactive Visuals*
- *Miscellaneous*
- *Reorganization*
- *Trace Processing*
- *Version*

3.1 Bruker

Write me
Write me
Write me
Write me

3.1.1 CalSciPy.bruker module

CalSciPy.bruker.**determine_imaging_content**(*folder*)

This function determines the number of channels and planes within a folder containing .tif files exported by Bruker's Prairieview software. It also determines the size of the images (frames, y-pixels, x-pixels). It's a quick / fast alternative to parsing its respective xml. Dependent on the naming conventions of PrairieView.

Parameters
folder (*str or pathlib.Path*) – folder containing bruker imaging data

Returns
channels, planes, frames, height, width

Return type
tuple[int, int, int, int, int]

CalSciPy.bruker.**generate_bruker_naming_convention**(*channel=0*, *plane=0*, *num_channels=1*, *num_planes=1*)

Generates the expected bruker naming convention for images collected with an arbitrary number of cycles & channels

This function expects that the naming convention is {experiment_name}_Cycle00000_Ch0_000000.ome.tiff where the channel is one-indexed. The 5-digit cycle id represents the frame if using multiplane imaging and the 6-digit tag represents the plane. Otherwise, the 5-digit tag is static and the 6-digit tag represents the frame. Channel and plane are *zero-indexed*.

Parameters

- **channel** (*int = 0*) – channel to produce name for
- **plane** (*int = 0*) – plane to produce name for
- **num_channels** (*int = 1*) – number of channels
- **num_planes** (*int = 1*) – number of planes

Return type
str

Returns

CalSciPy.bruker.**load_bruker_tifs**(*folder*, *channel=None*, *plane=None*)

Load images collected and exported to .tif by Bruker's Prairieview software to a tuple of numpy arrays. If multiple channels or multiple planes exist, each channel and plane combination is loaded to a separate numpy array. Identification of multiple channels / planes is dependent on *determine_imaging_content()*. Images are loaded as unsigned 16-bit, though note that raw bruker files are natively 12 or 13-bit.

Parameters

- **folder** (*str or pathlib.Path*) – folder containing a sequence of single frame tiff files
- **channel** (*Optional[int] = None*) – specific channel to load from dataset (zero-indexed)
- **plane** (*Optional[int] = None*) – specific plane to load from dataset (zero-indexed)

Returns
All .tif files in the directory loaded to a tuple of numpy arrays (frames, y-pixels, x-pixels, `np.uint16`)

Return type
tuple[numpy.ndarray]

CalSciPy.bruker.**repackage_bruker_tifs**(*input_folder*, *output_folder*, *channel=0*, *plane=0*)

Repackages a folder containing .tif files exported by Bruker's Prairieview software into a sequence of <4 GB .tif stacks. Channels are planes are **zero-indexed**.

Parameters

- **input_folder** (*str or pathlib.Path*) – folder containing a sequence of single frame .tif files exported by Bruker's Prairieview
- **output_folder** (*str or pathlib.Path*) – empty folder where .tif stacks will be saved
- **channel** (*int = 0*) – optional input specifying channel
- **plane** (*int = 0*) – optional input specifying plane

Return type
None

3.2 Coloring

Write me
Write me
Write me
Write me

3.2.1 Coloring Methods

Import me

3.3 Event Processing

Write me
Write me
Write me
Write me

3.3.1 CalSciPy.event_processing module

CalSciPy.event_processing.**calculate_firing_rates**(*spike_probability_matrix*, *frame_rate=30.0*, *in_place=False*)

Calculate firing rates

Parameters

- **spike_probability_matrix** (*numpy.ndarray*) – matrix of n neuron x m samples where each element is the probability of a spike
- **frame_rate** (*float = 30*) – frame rate of dataset
- **in_place** (*bool = False*) – boolean indicating whether to perform calculation in-place

Returns
firing matrix of n neurons x m samples where each element is a binary indicating presence of spike event

Return type
numpy.ndarray

CalSciPy.event_processing.**calculate_mean_firing_rates**(*firing_matrix*)

Calculate mean firing rate

Parameters
firing_matrix (*numpy.ndarray*) – matrix of n neuron x m samples where each element is either a spike or an instantaneous firing rate

Returns
1-D vector of mean firing rates

Return type
numpy.ndarray

CalSciPy.event_processing.**collect_waveforms**(*traces*, *event_indices*, *pre=150*, *post=450*)

Collect waveforms for each event

Parameters

- **traces** (*numpy.ndarray*) – a matrix of M neurons x N samples
- **event_indices** (*Iterable[Iterable[int]]*) – a list of events
- **pre** (*int*) – number of pre-event frames
- **post** (*int*) – number of post-event frames

Returns
a matrix of M events x N samples

Return type
Tuple[numpy.ndarray]

CalSciPy.event_processing.**convert_tau**(*tau*, *dt*)

Converts a discrete tau to a continuous tau

Parameters

- **tau** (*float*) – decay constant
- **dt** (*float*) – time step (s)

Returns
continuous tau (s)

Return type
float

CalSciPy.event_processing.**get_event_onset_intensities**(*traces*, *event_indices*)

Retrieve the signal intensity at event onset for each neuron in the event indices

Parameters

- **traces** (*numpy.ndarray*) – An M neuron by N sample matrix
- **event_indices** (*Iterable[Iterable[int]]*) – An iterable of length M containing a sequence with a duration for each event

Returns
An iterable of length M neurons containing the onset intensities for each event in the sequence

Return type
Tuple[numpy.ndarray]

CalSciPy.event_processing.**get_inter_event_intervals**(*event_indices*, *frame_rate=30.0*)

Calculate the inter event intervals for each neuron in the event indices

Parameters

- **event_indices** (*Iterable[Iterable[int]]*) – An iterable of length M containing a sequence with a duration for each event
- **frame_rate** (*float*) – frame_rate for trace matrix

Returns
An iterable of length M neurons containing the inter-event intervals for each event in the sequence

Return type
Tuple[numpy.ndarray]

CalSciPy.event_processing.**get_num_events**(*event_indices*)

Determines the number of events for each neuron in the event indices

Parameters
event_indices (*Iterable[Iterable[int]]*) – An iterable of length M neurons containing a sequence with a duration for each event

Returns
A 1-D vector of length M neurons containing the number of events for each neuron

Return type
numpy.ndarray

CalSciPy.event_processing.**identify_events**(*traces*, *timeout=15*, *frame_rate=30.0*, *smooth=True*, *force_nonneg=True*)

Identify event onset for each neuron using the smoothed, non-negative first-time derivative. The threshold for noise is considered 1/2th the standard deviation of the derivative.

Parameters

- **traces** (*numpy.ndarray*) – An M neuron by N sample matrix
- **timeout** (*int*) – timeout distance for peak finding (frames)
- **frame_rate** (*float*) – frame rate / time step for trace matrix
- **smooth** (*bool = True*) – boolean indicating whether to smooth first-time derivative
- **force_nonneg** (*bool = True*) – boolean indicating whether to enforce non-negativity constraint on first-time derivative

Returns
An iterable where each element contains a sequence of frames identified as event onsets

Return type
Tuple[List[int]]

CalSciPy.event_processing.**normalize_firing_rates**(*firing_matrix*, *in_place=False*)

Normalize firing rates by scaling to a max of 1.0. Non-negativity constrained.

Parameters

- **firing_matrix** (*numpy.ndarray*) – matrix of n neuron x m samples where each element is either a spike or an instantaneous firing rate
- **in_place** (*bool = False*) – boolean indicating whether to perform calculation in-place

Returns
normalized firing rate matrix of n neurons x m samples

Return type
numpy.ndarray

CalSciPy.event_processing.**scale_waveforms**(*waveforms*, *scaler=<class 'sklearn.preprocessing._data.StandardScaler'>*)

Scale waveforms for cross-neuron comparisons

Parameters

- **waveforms** (*numpy.ndarray*) – An Iterable of M events by N samples matrices of waveforms
- **scaler** (*Callable*) – sklearn preprocessing object

Returns
An Iterable of M event by N samples scaled matrices of waveforms

Return type
Iterable[numpy.ndarray]

3.4 Input/Output (I/O)

Write me
Write me
Write me
Write me

3.4.1 CalSciPy.io_tools module

CalSciPy.io_tools.**load_binary**(*path*, *mapped=False*)

Return type
Union[ndarray, memmap]

CalSciPy.io_tools.**load_images**(*path*)

Load images into a numpy array. If path is a folder, all .tif files found non-recursively in the directory will be compiled to a single array

Parameters
path (*str or pathlib.Path*) – a file containing images or a folder containing several imaging stacks

Returns
numpy array (frames, y-pixels, x-pixels)

Return type
numpy.ndarray

CalSciPy.io_tools.**save_binary**(*path*, *images*)

Save images to language-agnostic binary format. Ideal for optimal read/write speeds and highly-robust to corruption. However, the downside is that the images and their metadata are split into two separate files. Images are saved with the *.bin* extension, while metadata is saved with extension *.json*. If for some reason you lose the metadata, you can still load the binary if you know three of the following: number of frames, y-pixels, x-pixels, and the datatype. The datatype is almost always unsigned 16-bit for all modern imaging systems–even if they are collected at 12 or 13-bit.

Parameters
path (Union[str, Path]) – path to save images to. The path stem is considered the filename if it doesn't have any extension.

If no filename is provided then the default filename is *binary_video*. :type path: str or pathlib.Path :type images: ndarray :param images: images to save (frames, y-pixels, x-pixels) :type images: numpy.ndarray :return: 0 if successful :rtype: int

CalSciPy.io_tools.**save_images**(*path*, *images*, *size_cap=3.9*)

Save a numpy array to a single .tif file. If size > 4GB then saved as a series of files. If path is not a file and already exists the default filename will be *images*.

Parameters

- **path** (*str or pathlib.Path*) – filename or absolute path
- **images** (*numpy.ndarray*) – numpy array (frames, y pixels, x pixels)
- **size_cap** (*float = 3.9*) – maximum size per file

Returns
returns 0 if successful

Return type
int

3.5 Image Processing

Write me
Write me
Write me
Write me

3.5.1 CalSciPy.image_processing module

CalSciPy.image_processing.**gaussian_filter**(*images*, *sigma=1.0*, *block_size=None*, *block_buffer=0*, *in_place=False*)

GPU-parallelized multidimensional gaussian filter. Optional arguments for in-place calculation. Can be calculated blockwise with overlapping or non-overlapping blocks.

Designed for use on arrays larger than the available memory capacity.

Footprint is of the form np.ones((frames, y pixels, x pixels)) with the origin in the center

Parameters

- **images** (*numpy.ndarray*) – images stack to be filtered
- **sigma** (*Number or numpy.ndarray*) – sigma for gaussian filter
- **block_size** (*int = None*) – the size of each block. Must fit within memory
- **block_buffer** (*int = 0*) – the size of the overlapping region between block
- **in_place** (*bool = False*) – whether to calculate in-place

Returns
images: numpy array (frames, y pixels, x pixels)

Return type
numpy.ndarray

CalSciPy.image_processing.**median_filter**(*images*, *mask=array([[[1., 1., 1.], [1., 1., 1.], [1., 1., 1.]], [[1., 1., 1.], [1., 1., 1.], [1., 1., 1.]], [[1., 1., 1.], [1., 1., 1.], [1., 1., 1.]]])*, *block_size=None*, *block_buffer=0*, *in_place=False*)

GPU-parallelized multidimensional median filter. Optional arguments for in-place calculation. Can be calculated blockwise with overlapping or non-overlapping blocks.

Designed for use on arrays larger than the available memory capacity.

Footprint is of the form np.ones((frames, y pixels, x pixels)) with the origin in the center

Parameters

- **images** (*numpy.ndarray*) – images stack to be filtered
- **mask** (*numpy.ndarray = np.ones((3, 3, 3))*) – mask of the median filter
- **block_size** (*int = None*) – the size of each block. Must fit within memory
- **block_buffer** (*int = 0*) – the size of the overlapping region between block
- **in_place** (*bool = False*) – whether to calculate in-place

Returns
images: numpy array (frames, y pixels, x pixels)

Return type
numpy.ndarray

3.6 Interactive Visuals

Write me
Write me
Write me
Write me

3.6.1 Interactive Visuals Methods

Import me

Miscellaneous

Write me
Write me
Write me
Write me

3.6.2 Miscellaneous Methods

Import me

3.7 Reorganization

Write me
Write me
Write me
Write me

3.7.1 Reorganization Methods

CalSciPy.reorganization module

CalSciPy.reorganization.**generate_raster**(*event_frames*, *total_frames=None*)

Generate raster from an iterable of iterables containing the spike or event times for each neuron

Parameters

- **event_frames** (*Iterable[Iterable[int]]*) – iterable containing an iterable identifying the event frames for each neuron
- **total_frames** (*Optional[int] = None*) – total number of frames

Returns
event matrix of neurons x total frames

Return type
numpy.ndarray

CalSciPy.reorganization.**generate_tensor**(*traces_as_matrix*, *chunk_size*)

Generates a tensor given chunk / trial indices

Parameters

- **traces_as_matrix** (*numpy.ndarray*) – traces in matrix form (neurons x frames)
- **chunk_size** (*int*) – size of each chunk

Returns
traces_as_tensor

Return type
numpy.ndarray

CalSciPy.reorganization.**merge_factorized_matrices**(*factorized_traces*, *component=0*)

Concatenate a neuron x chunk or trial array in which each element is a component x frame factorization of the original trace:

Parameters

- **factorized_traces** (*numpy.ndarray*) – neurons x chunks (trial, tiff, etc) containing the neuron's trace factorized into several components
- **component** (*int*) – specific component to extract

Returns
traces of specific component in matrix form

Return type
numpy.ndarray

CalSciPy.reorganization.**merge_tensor**(*traces_as_tensor*)

Concatenate multiple trials or tiffs into single matrix:

Parameters
traces_as_tensor (*numpy.ndarray*) – chunk (trial, tiff, etc) x neurons x frames

Returns
traces in matrix form

Return type
numpy.ndarray

3.8 Trace Processing

Write me
Write me
Write me
Write me

3.8.1 CalSciPy.trace_processing module

CalSciPy.trace_processing.**calculate_dfof**(*traces*, *frame_rate=30.0*, *in_place=False*, *offset=0.0*, *external_reference=None*)

Calculates f/f0 (fold fluorescence over baseline). Baseline is defined as the 5th percentile of the signal after a 1Hz low-pass filter using a Hamming window. Baseline can be calculated using an external reference using the raw argument or adjusted by using the offset argument. Supports in-place calculation (off by default).

Parameters

- **traces** (*numpy.ndarray*) – matrix of traces in the form of neurons x frames
- **frame_rate** (*float = 30.0*) – frame rate of dataset
- **in_place** (*bool = False*) – boolean indicating whether to perform calculation in-place
- **offset** (*float = 0.0*) – offset added to baseline; useful if traces are non-negative
- **external_reference** (*numpy.ndarray = None*) – secondary dataset used to calculate baseline; useful if traces have been factorized

Returns
f/f0 matrix of n neurons x m samples

Return type
numpy.ndarray

CalSciPy.trace_processing.**calculate_standardized_noise**(*fold_fluorescence_over_baseline*, *frame_rate=30.0*)

Calculates a frame-rate independent standardized noise as defined as:

$$v = \frac{\sigma \frac{\Delta F}{F}}{\sqrt{f}}$$

It is robust against outliers and approximates the standard deviation of f/f0 baseline fluctuations. For comparison, the more exquisite of the Allen Brain Institute's public datasets are approximately 1*%Hz^(-1/2)

Parameters

- **fold_fluorescence_over_baseline** (*numpy.ndarray*) – fold fluorescence over baseline (i.e., f/f0)
- **frame_rate** (*float = 30*) – frame rate of dataset

Returns
standardized noise (units are 1*%Hz^(-1/2)) for each neuron

Return type
numpy.ndarray

CalSciPy.trace_processing.**detrend_polynomial**(*traces*, *in_place=False*)

Detrend traces using a fourth-order polynomial

Parameters

- **traces** (*numpy.ndarray*) – matrix of traces in the form of neurons x frames
- **in_place** (*bool = False*) – boolean indicating whether to perform calculation in-place

Returns
detrended traces

Return type
numpy.ndarray

CalSciPy.trace_processing.**perona_malik_diffusion**(*traces*, *iters=25*, *kappa=0.15*, *gamma=0.25*, *in_place=False*)

Edge-preserving smoothing using perona malik diffusion. This is a non-linear smoothing technique that avoids the temporal distortion introduced onto traces by standard gaussian smoothing.

The parameter *kappa* controls the level of smoothing ("diffusion") as a function of the derivative of the trace (or "gradient" in the case of 2D images where this algorithm is often used). This function is known as the diffusion coefficient. When the derivative for some portion of the trace is low, the algorithm will encourage smoothing to reduce noise. If the derivative is large like during a burst of activity, the algorithm will discourage smoothing to maintain its structure. Here, the argument *kappa* is multiplied by the dynamic range to generate the true kappa.

represents the percentile used to calculate the true kappa

The diffusion coefficient implemented here is e^(-(derivative/kappa)^2).

Perona-Malik diffusion is an iterative process. The parameter *gamma* controls the rate of diffusion, while parameter *iters* sets the number of iterations to perform.

This implementation is currently situated to handle 1-D vectors because it gives us some performance benefits.

Parameters

- **traces** (*numpy.ndarray*) – matrix of M neurons by N samples
- **iters** (*int = 25*) – number of iterations
- **kappa** (*Number = 15*) – used to calculate the true kappa, where true kappa = kappa * dynamic range. range 0-1
- **gamma** (*float = 0.25*) – rate of diffusion for each iter. range 0-1
- **in_place** (*bool = False*) – whether to calculate in-place

Returns
smoothed traces

Return type
numpy.ndarray

Write me
Write me
Write me
Write me

3.8.2 Version Methods

Import Me

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C