CalSciPy

Release 0.0.5

Darik A. O'Neil

Jan 24, 2023

Contents:

CHAPTER 1

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 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. When each module has its associated unit tests complete, it will be pushed.

CHAPTER 2

Installation

2.1 Full Install

Enter `pip install CalSciPy` in your terminal.

2.2 Partial Install

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

CHAPTER 3

Overview

3.1 Coloring

Write me
Write me
Write me
Write me

3.1.1 Coloring Methods

Import me

3.2 Event Processing

Write me
Write me
Write me
Write me

3.2.1 Event Processing Methods

Import me

3.3 Input/Output (I/O)

Write me
Write me
Write me
Write me

3.3.1 CalSciPy.io module

CalSciPy.io.**determine_bruker_folder_contents**(*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)

Parameters
folder (*Union[str, pathlib.Path]*) -- folder containing bruker imaging data

Returns
channels, planes, frames, height, width

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

CalSciPy.io.**load_all_tiffs**(*folder*)

Loads all .tif's within a folder into a single numpy array. Assumes .tif files are the standard unsigned 16-bit exported by the majority (all?) of imaging software.

Parameters
folder (*Union[str, pathlib.Path]*) -- folder containing a sequence of tiff stacks

Returns
a numpy array containing the images (frames, y-pixels, x-pixels)

Return type
numpy.ndarray

CalSciPy.io.**load_binary_meta**(*path*)

Loads the meta file for an associated binary video

Parameters
path (*Union[str, pathlib.Path]*) -- The meta file (.txt ext) or a directory containing metafile

Returns
A tuple where (frames, y-pixels, x-pixels, dtype)

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

CalSciPy.io.**load_bruker_tiffs**(*folder*, *channels=None*, *planes=None*)

Load a sequence of .tif files from a directory containing .tif files exported by Bruker's Prairieview software to a numpy array. If multiple channels or multiple planes exist, each channel and plane combination is loaded to a separate numpy array.

Parameters

- **folder** (*Union[str, pathlib.Path]*) -- folder containing a sequence of single frame tiff files
- **channels** (*Optional[int]*) -- specific channel to load from dataset (zero-indexed)
- **planes** (*Optional[int]*) -- 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, uint16)

Return type
tuple[numpy.ndarray]

CalSciPy.io.**load_mapped_binary**(*path*, *meta_filename*, ***kwargs*)

Loads a raw binary file as numpy array without loading into memory (memmap). Enter a directory to autogenerate the default filenames (binary_video, video_meta.txt)

Parameters

- **path** (*str*) -- absolute filepath for binary video or a folder containing a binary video with the default filename
- **meta_filename** (*Optional[str] = None*) -- absolute path to meta file
- **mode** -- mode used in loading numpy.memmap (str, default = "r")

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

Return type
numpy.memmap

CalSciPy.io.**load_raw_binary**(*path*, *meta_filename*)

Loads a raw binary file as a numpy array. Enter a directory to autogenerate the default filenames (binary_video, video_meta.txt)

Parameters

- **path** (*str*) -- absolute filepath for binary video or directory containing a file named binary video
- **meta_filename** (*Optional[str] = None*) -- absolute path to meta file

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

Return type
numpy.ndarray

CalSciPy.io.**load_single_tiff**(*path*, *num_frames*)

Load a single .tif as a numpy array

Parameters

- **path** (*Union[str, pathlib.Path]*) -- absolute filename
- **num_frames** (*int*) -- number of frames in .tif

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

Return type
numpy.ndarray

CalSciPy.io.**pretty_print_image_description**(*channels*, *planes*, *frames*, *height*, *width*)

Function prints the description of an imaging dataset as a table.

Parameters

- **channels** (*int*) -- number of channels
- **planes** (*int*) -- number of planes
- **frames** (*int*) -- number of frames
- **height** (*int*) -- y-pixels
- **width** -- x-pixels

Return type
None

CalSciPy.io.**repackage_bruker_tiffs**(*input_folder*, *output_folder*, **args*)

Repackages a folder containing .tif files exported by Bruker's Prairieview software into a sequence of <4 GB .tif stacks.

Parameters

- **input_folder** (*Union[str, pathlib.Path]*) -- folder containing a sequence of single frame .tif files exported by Bruker's Prairieview
- **output_folder** (*Union[str, pathlib.Path]*) -- empty folder where .tif stacks will be saved
- **args** (*int*) -- optional argument to indicate the repackaging of a specific channel and/or plane

Return type
None

CalSciPy.io.**save_raw_binary**(*images*, *path*, *meta_filename*)

Save a numpy array as a binary file with an associated meta .txt file

Parameters

- **images** (*numpy.ndarray*) -- numpy array (frames, y-pixels, x-pixels)
- **path** (*str*) -- folder to save in or an absolute filepath for binary video file

- **meta_filename** (*str*) -- absolute filepath for saving meta .txt file

Return type
None

CalSciPy.io.**save_single_tiff**(*images*, *path*, *type_=<class 'numpy.uint16'>*)

Save a numpy array to a single .tif file. Size must be <4 GB.

Parameters

- **images** (*numpy.array*) -- numpy array [frames, y pixels, x pixels]
- **path** (*Union[str, pathlib.Path]*) -- filename or absolute path
- **type** (*Optional[numpy.dtype] = numpy.uint16*) -- type for saving

Return type
None

CalSciPy.io.**save_tiff_stack**(*images*, *output_folder*, *type_=<class 'numpy.uint16'>*)

Save a numpy array to a sequence of .tif stacks

Parameters

- **images** (*numpy.array*) -- numpy array (frames, y-pixels, x-pixels)
- **output_folder** (*Union[str, pathlib.Path]*) -- folder to save the sequence of .tif stacks
- **type** (*Optional[numpy.dtype] = numpy.uint16*) -- type for saving

Return type
None

CalSciPy.io.**save_video**(*images*, *path*, *fps=30*)

Save numpy array as an .mp4. Will be converted to uint8 if any other datatype.

Parameters

- **images** (*numpy.array*) -- numpy array (frames, y-pixels, x-pixels)
- **path** (*Union[str, pathlib.Path]*) -- absolute filepath or filename
- **fps** (*Union[float, int] = 30*) -- frame rate for saved video

Return type
None

3.4 Image Processing

Write me
Write me
Write me
Write me

3.5 Interactive Visuals

Write me
Write me
Write me
Write me

3.5.1 Interactive Visuals Methods

Import me

3.6 Reorganization

Write me
Write me
Write me
Write me

3.6.1 Reorganization Methods

Import me

3.7 Signal Processing

Write me
Write me
Write me
Write me

3.7.1 Signal Processing Methods

Import me

3.8 Static Visuals

Write me
Write me
Write me
Write me

3.8.1 Static Visual Methods

Import me

CHAPTER 4

Development Tools

write me
write me
write me
write me

4.1 Parameterized Decorators

write me
write me
write me
write me

4.1.1 Parsing Decorators

write me
write me
write me
write me

4.1.2 Validation Decorators

write me
write me
write me
write me

4.1.3 Terminal Style

write me
write me
write me
write me

4.2 PyTest

write me
write me
write me
write me

4.2.1 Sample Datasets

write me
write me
write me
write me

4.2.2 Tests

write me
write me
write me
write me

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

C