CalSciPy

Release 0.0.5

Darik A. O'Neil

Jan 23, 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.

1.3 Installation

Enter `pip install CalSciPy` in your terminal.

CHAPTER 2

Modules

2.1 Coloring

2.1.1 Description

Write me
Write me
Write me
Write me

2.1.2 Methods

Import me

2.2 Event Processing

2.2.1 Description

Write me
Write me
Write me
Write me

2.2.2 Methods

Import me

2.3 Input/Output (I/O)

2.3.1 Description

Write me
Write me
Write me
Write me

2.3.2 Methods

Import me

2.4 Image Processing

2.4.1 Description

Write me
Write me
Write me
Write me

2.4.2 Methods

Import me

2.5 Interactive Visuals

2.5.1 Description

Write me
Write me
Write me
Write me

2.5.2 Methods

Import me

2.6 Reorganization

2.6.1 Description

Write me
Write me
Write me
Write me

2.6.2 Methods

Import me

2.7 Signal Processing

2.7.1 Description

Write me
Write me
Write me
Write me

2.7.2 Methods

Import me

2.8 Static Visuals

2.8.1 Description

Write me
Write me
Write me
Write me

2.8.2 Methods

Import me

CHAPTER 3

CalSciPy package

3.1 Submodules

3.1.1 CalSciPy.coloring module

3.1.2 CalSciPy.event_processing module

3.1.3 CalSciPy.image_processing module

3.1.4 CalSciPy.interactive_visuals module

3.1.5 CalSciPy.io module

`CalSciPy.io.`**`determine_bruker_folder_contents`**`(`*folder*`)`

Function determine contents of the bruker folder

Parameters

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

Returns

channels, planes, frames, Height, Width

Return type

tuple

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

Load a sequence of tiff stacks

Parameters

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

Returns

complete_image numpy array [Z x Y x X] as uint16

Return type
np.ndarray

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

Loads meta file for binary video

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

Returns
A tuple containing the number of frames, y pixels, and x pixels [Z x Y x X]

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

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

Load a sequence of tiff files from a directory.

Designed to compile the outputs of a certain imaging utility that exports recordings such that each frame is saved as a single tiff.

Parameters

- **folder** (*Union[str, pathlib.Path]*) -- Folder containing a sequence of single frame tiff files
- **channels** (*Optional[int]*) -- channel to load
- **planes** (*Optional[int]*) -- plane to load

Returns
complete_image: All tiff files in the directory compiled into a single array (Z x Y x X, uint16)

Return type
Tuple[np.ndarray]

`CalSciPy.io.`**`load_mapped_binary`**`(`*filename*, *meta_filename*, ***kwargs*`)`

Loads a raw binary file in the workspace without loading into memory

Enter the path to autofill (assumes Filename & meta are path + binary_video, video_meta.txt)

Parameters

- **filename** (*str*) -- filename for binary video
- **meta_filename** (*str*) -- filename for meta file
- **mode** -- pass mode to numpy.memmap (str, default = "r")

Returns
memmap(numpy) array [Z x Y x X]

Return type
np.memmap

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

Loads a raw binary file

Enter the path to autofill (assumes Filename & meta are path + 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]*) -- absolute path to meta file

Returns
numpy array [Z x Y x X]

Return type
Any

`CalSciPy.io.`**`load_single_tiff`**(*filename*, *num_frames*)

Load a single tiff file

Parameters

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

Returns
numpy array [Z x Y x X]

Return type
np.ndarray

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

Function simply prints the bruker folder contents detected

Parameters

- **`channels`** (*int*) -- Number of channels
- **`planes`** (*int*) -- Number of planes
- **`frames`** (*int*) -- Number of frames
- **`height`** (*int*) -- Height of Image (Y Pixels)
- **`width`** -- Width of Image (X Pixels)

Return type
None

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

Repackages a sequence of tiff files within a directory to a smaller sequence of tiff stacks. Designed to compile the outputs of a certain imaging utility that exports recordings such that each frame is saved as a single tiff.

Parameters

- **`input_folder`** (*Union[str, pathlib.Path]*) -- Directory containing a sequence of single frame tiff files
- **`output_folder`** (*Union[str, pathlib.Path]*) -- Empty directory where tiff 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*)

This function saves a tiff stack as a binary file

Parameters

- **`images`** (*np.ndarray*) -- Images to be saved [Z x Y x X]

- **path** (*str*) -- absolute filepath for saving binary video or directory containing a file named binary video
- **meta_filename** (*str*) -- absolute filepath for saving meta

Return type
None

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

Save a numpy array to a single tiff file as type uint16

Parameters

- **images** (*Any*) -- numpy array [frames, y pixels, x pixels]
- **path** (*Union[str, pathlib.Path]*) -- filename or absolute path
- **type** (*Optional[Any]*) -- 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 tiff stacks

Parameters

- **images** (*Any*) -- A numpy array containing a tiff stack [Z x Y x X]
- **output_folder** (*Union[str, pathlib.Path]*) -- A directory to save the sequence of tiff stacks in uint16
- **type** (*Optional[Any]*) -- type for saving

Return type
None

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

Function writes video to .mp4

Parameters

- **images** (*Any*) -- Images to be written
- **path** (*Union[str, pathlib.Path]*) -- Filename (Or Complete Path)
- **fps** (*Union[float, int]*) -- frame rate

Return type
None

3.1.6 CalSciPy.reorganization module

`CalSciPy.reorganization.`**`merge_traces`**`(`*traces_as_tensor*`)`

Concatenate multiple trials or tiffs into single matrix:

Parameters
traces_as_tensor (ndarray) --

Returns

Return type
np.ndarray

3.1.7 CalSciPy.static_visuals module

3.1.8 CalSciPy.trace_processing module

3.2 Module contents

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