FISSA Documentation

Release 0.7.0

Sander Keemink
Scott Lowe

May 04, 2020

Contents:

FISSA (Fast Image Signal Separation Analysis) is a Python package for decontaminating somatic signals from two-photon calcium imaging data. It can read images in tiff format and ROIs from zip files exported by ImageJ; or operate on numpy arrays, generated by importing files stored in other or as the output of other packages.

For details of the algorithm, please see our companion paper published in Scientific Reports.

CHAPTER 1

FISSA User Guide

FISSA (Fast Image Signal Separation Analysis) is a Python package for decontaminating somatic signals from two-photon calcium imaging data. It can read images in tiff format and ROIs from zip files exported by ImageJ; or operate on numpy arrays, generated by importing files stored in other or as the output of other packages.

For details of the algorithm, please see our companion paper published in Scientific Reports. For the code used to generate the simulated data in the companion paper, see the SimCalc repository.

FISSA is compatible with both Python 2.7 and Python >=3.5, however Python 3 is strongly encouraged as Python 2 has reached its end of life. FISSA is continually tested on Ubuntu, Windows, and Mac OSX during its development cycle.

Documentation, including the full API, is available online at readthedocs.

If you encounter a specific problem please open a new issue. For general discussion and help with installation or setup, please see the Gitter chat.

1.1 Usage

A concise example of how to use FISSA is as follows.

```
import fissa

experiment = fissa.Experiment('path/to/tiffs', 'path/to/rois.zip', 'experiment_name')
experiment.separate()

# The separated time series data is now available as experiment.result
experiment.result[roi_index, tiff_index][0, :]
```

We also have several example notebooks for a basic worflow and more complicated workflows where FISSA needs to interact with the outputs of other two-photon calcium imaging toolboxes which can be used to automatically detect cells.

You can try out each of the example notebooks interactively in your browser on Binder (note that it may take 10 minutes for Binder to boot up).

Workflow	Jupyter Notebook			Script	
Basic (ImageJ)	View HTML	Launch Binder	Download	Linux/Mac	Windows
With suite2p	View HTML	Launch Binder	Download		
With SIMA	View HTML	Launch Binder	Download		
With CNMF (MATLAB)	View HTML	Launch Binder	Download		

These notebooks can also be run on your own machine. To do so, you will need to:

0. If you want to run the Suite2p notebook, you'll have to install everything into a conda environment, as per their installation instructions.
1. Install fissa with its plotting dependencies `pip install fissa[plotting]`.
2. If you want to run the sima notebook, you will also have to install sima with `pip install sima`. Note that sima only supports python<=3.6.
3. Download a copy of the repository, unzip it and browse to the examples directory.
4. Start up a jupyter notebook server to run our notebooks `jupyter notebook`.

If you're new to Jupyter notebooks, here is an approachable tutorial.

1.2 Installation

1.2.1 Quick Guide

FISSA is available on PyPI and the latest version can be installed into your current environment using pip.

```
pip install fissa
```

If you need more details or you're stuck with something in the dependency chain, more detailed instructions for both Windows and Ubuntu users are below.

1.2.2 Installation on Windows

We detail two different ways to install Python on your Windows. One is to download the *official Python installer*, and the other is to use *Anaconda*.

Official Python distribution

1. Go to the Python website and download the latest version of Python for Windows.
3. Run the executable file downloaded, which has a name formatted like **python-3.y.z.exe**.
4. In the installation window, tick the checkbox "Add Python 3.y to PATH".
5. Click "Install Now", and go though the installation process to install Python.
6. Open the **Command Prompt** application. We can run Python from the general purpose command prompt because we added its binaries to the global `PATH` variable in Step 4.
7. At the **Command Prompt** command prompt, install fissa and its dependencies by running the command:

   ```
   pip install fissa
   ```

8. You can check to see if fissa is installed with:

```
python -c "import fissa; print(fissa.__version__)"
```

   You should see your FISSA version number printed in the terminal.

9. You can now use FISSA from the Python command prompt. To open a python command prompt, either execute the command `python` within the **Command Prompt**, or open Python executable which was installed in Step 5. At the python command prompt, you can run FISSA as described in *Usage* above.

Anaconda distribution

1. Download and install the latest version of either Anaconda or Miniconda. Miniconda is a lightweight version of Anaconda, the same thing but without any packages pre-installed.
2. Open the **Anaconda Prompt**, which was installed by either Anaconda or Miniconda in Step 1.
3. In the Anaconda Prompt, run the following command to install some of FISSA's dependencies with conda.

```
conda install -c conda-forge numpy scipy shapely tifffile
```

4. Run the following command to install FISSA, along with the rest of its dependencies.

```
pip install fissa
```

5. You can check to see if fissa is installed with:

```
python -c "import fissa; print(fissa.__version__)"
```

   You should see your FISSA version number printed in the terminal.

6. You can now use FISSA from the Python command prompt. To open a python command prompt, either execute the command `python` within the **Anaconda Prompt**. At the python command prompt, you can run FISSA as described in *Usage* above.
7. Optionally, if you want use suite2p, it and its dependencies can be installed as follows.

```
conda install -c conda-forge mkl mkl_fft numba pyqt
pip install suite2p rastermap
```

1.2.3 Installation on Linux

Before installing FISSA, you will need to make sure you have all of its dependencies (and the dependencies of its dependencies) installed.

Here we will outline how to do all of these steps, assuming you already have both Python and pip installed. It is highly likely that your Linux distribution ships with these. You can upgrade to a newer version of Python by downloading Python from the official website.

Alternatively, you can use an *Anaconda* environment (not detailed here).

1. Dependencies of dependencies
   - scipy requires a Fortran compiler and BLAS/LAPACK/ATLAS
   - shapely requires GEOS.
   - Pillow>=3.0.0 effectively requires a JPEG library.

These packages can be installed on Debian/Ubuntu with the following shell commands.

```
sudo apt-get update
sudo apt-get install gfortran libopenblas-dev liblapack-dev libatlas-dev libatlas-
↪base-dev
sudo apt-get install libgeos-dev
sudo apt-get install libjpeg-dev
```

2. Install the latest release version of FISSA from PyPI using pip:

```
pip install fissa
```

3. You can check to see if FISSA is installed by running the command:

```
python -c "import fissa; print(fissa.__version__)"
```

You will see your FISSA version number printed in the terminal.

4. You can now use FISSA from the Python command prompt, as described in *Usage* above.

1.3 Citing FISSA

If you use FISSA for your research, we would be grateful if you could cite our paper on FISSA in any resulting publications:

> S. W. Keemink, S. C. Lowe, J. M. P. Pakan, E. Dylda, M. C. W. van Rossum, and N. L. Rochefort. FISSA: A neuropil decontamination toolbox for calcium imaging signals, *Scientific Reports*, **8**(1):3493, 2018. doi: 10.1038/s41598-018-21640-2.

For your convenience, we provide a copy of this citation in bibtex and RIS format.

1.4 License

CHAPTER 2

API Reference

2.1 fissa package

2.1.1 Submodules

fissa.ROI module

The functions below were adapted from the sima package http://www.losonczylab.org/sima version 1.3.0.

License

`fissa.ROI.`**`poly2mask`**(*polygons*, *im_size*)

Converts polygons to a sparse binary mask.

```
>>> from fissa.ROI import poly2mask
>>> poly1 = [[0,0], [0,1], [1,1], [1,0]]
>>> poly2 = [[0,1], [0,2], [2,2], [2,1]]
>>> mask = poly2mask([poly1, poly2], (3, 3))
>>> mask[0].todense()
matrix([[ True, False, False],
```

(continues on next page)

(continued from previous page)

```
        [ True,  True, False],
        [False, False, False]], dtype=bool)
```

Parameters

- **polygons** (*sequence of coordinates or sequence of Polygons*) – A sequence of polygons where each is either a sequence of (x,y) or (x,y,z) coordinate pairs, an Nx2 or Nx3 numpy array, or a Polygon object.
- **im_size** (*tuple*) – Final size of the resulting mask

Returns masks – A list of sparse binary masks of the points contained within the polygons, one mask per plane. Each mask is in linked list sparse matrix format.

Return type list of sparse matrices

fissa.core module

Main user interface for FISSA.

Authors:

- Sander W Keemink (swkeemink@scimail.eu)
- Scott C Lowe

class fissa.core.**Experiment**(*images*, *rois*, *folder*, *nRegions=4*, *expansion=1*, *alpha=0.1*, *ncores_preparation=None*, *ncores_separation=None*, *method='nmf'*, *lowmemory_mode=False*, *datahandler_custom=None*)

Bases: object

Does all the steps for FISSA.

__init__(*images*, *rois*, *folder*, *nRegions=4*, *expansion=1*, *alpha=0.1*, *ncores_preparation=None*, *ncores_separation=None*, *method='nmf'*, *lowmemory_mode=False*, *datahandler_custom=None*)

Initialisation. Set the parameters for your Fissa instance.

Parameters

- **images** (*str or list*) – The raw recording data. Should be one of:
  - the path to a directory containing TIFF files (string),
  - an explicit list of TIFF files (list of strings),
  - a list of array_like data already loaded into memory, each shaped *(frames, y-coords, x-coords)*.

  Note that each TIFF/array is considered a single trial.
- **rois** (*str or list*) – The roi definitions. Should be one of:
  - the path to a directory containing ImageJ ZIP files (string),
  - the path of a single ImageJ ZIP file (string),
  - a list of ImageJ ZIP files (list of strings),
  - a list of arrays, each encoding a ROI polygons,
  - a list of lists of binary arrays, each representing a ROI mask.

This can either be a single roiset for all trials, or a different roiset for each trial.

- **folder** (*str*) – Output path to a directory in which the extracted data will be stored.
- **nRegions** (*int, optional*) – Number of neuropil regions to draw. Use a higher number for densely labelled tissue. Default is 4.
- **expansion** (*float, optional*) – Expansion factor for the neuropil region, relative to the ROI area. Default is 1. The total neuropil area will be *nRegions * expansion * area(ROI)*.
- **alpha** (*float, optional*) – Sparsity regularizaton weight for NMF algorithm. Set to zero to remove regularization. Default is 0.1. (Not used for ICA method.)
- **ncores_preparation** (*int, optional (default: None)*) – Sets the number of subprocesses to be used during the data preparation steps (ROI and subregions definitions, data extraction from tifs, etc.). If set to *None* (default), there will be as many subprocesses as there are threads or cores on the machine. Note that this behaviour can, especially for the data preparation step, be very memory-intensive.
- **ncores_separation** (*int, optional (default: None)*) – Same as *ncores_preparation*, but for the separation step. Note that this step requires less memory per subprocess, and hence can often be set higher than *ncores_preparation*.
- **method** (*{'nmf', 'ica'}, optional*) – Which blind source-separation method to use. Either *'nmf'* for non-negative matrix factorization, or *'ica'* for independent component analysis. Default (recommended) is *'nmf'*.
- **lowmemory_mode** (*bool, optional*) – If *True*, FISSA will load TIFF files into memory frame-by-frame instead of holding the entire TIFF in memory at once. This option reduces the memory load, and may be necessary for very large inputs. Default is *False*.
- **datahandler_custom** (*object, optional*) – A custom datahandler for handling ROIs and calcium data can be given here. See datahandler.py (the default handler) for an example.

calc_deltaf(*freq*, *use_raw_f0=True*, *across_trials=True*)

Calculate deltaf/f0 for raw and result traces.

The results can be accessed as self.deltaf_raw and self.deltaf_result. self.deltaf_raw is only the ROI trace instead of the traces across all subregions.

Parameters

- **freq** (*float*) – Imaging frequency, in Hz.
- **use_raw_f0** (*bool, optional*) – If *True* (default), use an f0 estimate from the raw ROI trace for both raw and result traces. If *False*, use individual f0 estimates for each of the traces.
- **across_trials** (*bool, optional*) – If *True*, we estimate a single baseline f0 value across all trials. If *False*, each trial will have their own baseline f0, and df/f0 value will be relative to the trial-specific f0. Default is *True*.

save_to_matlab()

Save the results to a matlab file.

Can be found in *folder/matlab.mat*.

This will give you a filename.mat file which if loaded in Matlab gives the following structs: ROIs, result, raw.

If df/f0 was calculated, these will also be stored as *df_result* and *df_raw*, which will have the same format as result and raw.

These can be interfaced with as follows, for cell 0, trial 0:

- *ROIs.cell0.trial0{1}* polygon for the ROI
- *ROIs.cell0.trial0{2}* polygon for first neuropil region
- *result.cell0.trial0(1,:)* final extracted cell signal
- *result.cell0.trial0(2,:)* contaminating signal
- *raw.cell0.trial0(1,:)* raw measured celll signal
- *raw.cell0.trial0(2,:)* raw signal from first neuropil region

separate(*redo_prep=False*, *redo_sep=False*)

Separate all the trials with FISSA algorithm.

After running *separate*, data can be found as follows:

self.sep Raw separation output, without being matched. Signal *i* for a specific cell and trial can be found as *self.sep[cell][trial][i,:]*.

self.result Final output, in order of presence in cell ROI. Signal *i* for a specific cell and trial can be found at *self.result[cell][trial][i, :]*. Note that the ordering is such that *i = 0* is the signal most strongly present in the ROI, and subsequent entries are in diminishing order.

self.mixmat The mixing matrix (how to go between *self.separated* and *self.raw* from the *separation_prep()* function).

self.info Information about separation routine, iterations needed, etc.

Parameters

- **redo_prep** (*bool, optional*) – Whether to redo the preparation. Default is *False.* Note that if this is true, we set *redo_sep = True* as well.
- **redo_sep** (*bool, optional*) – Whether to redo the separation. Default is *False*. Note that this parameter is ignored if *redo_prep* is set to *True*.

separation_prep(*redo=False*)

Prepare and extract the data to be separated.

For each trial, performs the following steps:

- Load in data as arrays
- Load in ROIs as masks
- Grow and seaparate ROIs to define neuropil regions
- Using neuropil and original ROI regions, extract traces from data

After running this you can access the raw data (i.e. pre-separation) as *self.raw* and *self.rois*. self.raw is a list of arrays. *self.raw[cell][trial]* gives you the traces of a specific cell and trial, across cell and neuropil regions. *self.roi_polys* is a list of lists of arrays. *self.roi_polys[cell][trial][region][0]* gives you the polygon for the region for a specific cell, trial and region. *region=0* is the cell, and *region>0* gives the different neuropil regions. For separateable masks, it is possible multiple outlines are found, which can be accessed as *self.roi_polys[cell][trial][region][i]*, where *i* is the outline index.

Parameters **redo** (*bool, optional*) – If *False*, we load previously prepared data when possible. If *True*, we re-run the preparation, even if it has previously been run. Default is *False*.

fissa.core.**extract_func**(*inputs*)
Extract data using multiprocessing.

Parameters **inputs** (*list*) – list of inputs

0. image array
1. the rois
2. number of neuropil regions
3. how much larger neuropil region should be then central ROI

Returns

- *dict* – Data across cells.
- *dict* – Polygons for each ROI.

fissa.core.**separate_func**(*inputs*)
Extraction function for multiprocessing.

Parameters **inputs** (*list*) – list of inputs

0. Array with signals to separate
1. Alpha input to npil.separate
2. Method
3. Current ROI number

Returns

- *numpy.ndarray* – The raw separated traces.
- *numpy.ndarray* – The separated traces matched to the primary signal.
- *numpy.ndarray* – Mixing matrix.
- *dict* – Metadata for the convergence result.

fissa.datahandler module

FISSA functions to handle image and roi objects and return the right format.

If a custom version of this file is used (which can be defined at the declaration of the core FISSA Experiment class), it should have the same functions as here, with the same inputs and outputs.

Authors:

- Sander W Keemink <swkeemink@scimail.eu>
- Scott C Lowe <scott.code.lowe@gmail.com>

fissa.datahandler.**extracttraces**(*data*, *masks*)
Extracts a temporal trace for each spatial mask.

Parameters

- **data** (*array_like*) – Data array as made by image2array. Should be shaped *(frames, y, x)*.
- **masks** (*list of array_like*) – List of binary arrays.

Returns Trace for each mask. Shaped *(len(masks), n_frames)*.

Return type numpy.ndarray

fissa.datahandler.**getmean**(*data*)
Determine the mean image across all frames.

Parameters **data** (*array_like*) – Data array as made by image2array. Should be shaped *(frames, y, x)*.

Returns y by x array for the mean values

Return type numpy.ndarray

fissa.datahandler.**image2array**(*image*)
Loads a TIFF image from disk.

Parameters **image** (*str or array_like*) – Either a path to a TIFF file, or array_like data.

Returns A 3D array containing the data, with dimensions corresponding to *(frames, y_coordinate, x_coordinate)*.

Return type numpy.ndarray

fissa.datahandler.**rois2masks**(*rois*, *data*)
Take the object *rois* and returns it as a list of binary masks.

Parameters

- **rois** (*string or list of array_like*) – Either a string with imagej roi zip location, list of arrays encoding polygons, or list of binary arrays representing masks
- **data** (*array*) – Data array as made by image2array. Must be shaped *(frames, y, x)*.

Returns List of binary arrays (i.e. masks)

Return type list

fissa.datahandler_framebyframe module

FISSA functions to handle image and roi objects and return the right format.

If a custom version of this file is used (which can be defined at the declaration of the core FISSA Experiment class), it should have the same functions as here, with the same inputs and outputs.

Authors:

- Sander W Keemink <swkeemink@scimail.eu>
- Scott C Lowe <scott.code.lowe@gmail.com>

fissa.datahandler_framebyframe.**extracttraces**(*data*, *masks*)
Get the traces for each mask in masks from data.

Parameters

- **data** (*PIL.Image*) – An open PIL.Image handle to a multi-frame TIFF image.
- **masks** (*list of array_like*) – List of binary arrays.

Returns Trace for each mask. Shaped *(len(masks), n_frames)*.

Return type numpy.ndarray

fissa.datahandler_framebyframe.**getmean**(*data*)
Determine the mean image across all frames.

Parameters **data** (*PIL.Image*) – An open PIL.Image handle to a multi-frame TIFF image.

Returns y-by-x array for the mean values.

Return type numpy.ndarray

fissa.datahandler_framebyframe.**image2array**(*image*)
Open a given image file as a PIL.Image instance.

Parameters **image** (*str or file*) – A filename (string) of a TIFF image file, a pathlib.Path object, or a file object.

Returns Handle from which frames can be loaded.

Return type PIL.Image

fissa.datahandler_framebyframe.**rois2masks**(*rois*, *data*)
Take the object 'rois' and returns it as a list of binary masks.

Parameters

- **rois** (*str or list of array_like*) – Either a string with imagej roi zip location, list of arrays encoding polygons, or list of binary arrays representing masks
- **data** (*PIL.Image*) – An open PIL.Image handle to a multi-frame TIFF image.

Returns List of binary arrays (i.e. masks).

Return type list

fissa.deltaf module

Functions for computing correcting fluorescence signals for changes in baseline activity.

Authors:

- Scott C Lowe

fissa.deltaf.**findBaselineF0**(*rawF*, *fs*, *axis=0*, *keepdims=False*)
Find the baseline for a fluorescence imaging trace line.

The baseline, F0, is the 5th-percentile of the 1Hz lowpass filtered signal.

Parameters

- **rawF** (*array_like*) – Raw fluorescence signal.
- **fs** (*float*) – Sampling frequency of rawF, in Hz.
- **axis** (*int, optional*) – Dimension which contains the time series. Default is 0.
- **keepdims** (*bool, optional*) – Whether to preserve the dimensionality of the input. Default is *False*.

Returns **baselineF0** – The baseline fluorescence of each recording, as an array.

Return type numpy.ndarray

Note: In typical usage, the input rawF is expected to be sized *(numROI, numTimePoints, numRecs)* and the output will then be sized *(numROI, 1, numRecs)* if *keepdims* is *True*.

---

fissa.neuropil module

Functions for removal of neuropil from calcium signals.

Authors:

- Sander W Keemink (swkeemink@scimail.eu)
- Scott C Lowe

Created: 2015-05-15

fissa.neuropil.**lowPassFilter**(*F*, *fs=40*, *nfilt=40*, *fw_base=10*, *axis=0*)

Low pass filters a fluorescence imaging trace line.

Parameters

- **F** (*array_like*) – Fluorescence signal.
- **fs** (*float, optional*) – Sampling frequency of F, in Hz. Default is 40.
- **nfilt** (*int, optional*) – Number of taps to use in FIR filter. Default is 40.
- **fw_base** (*float, optional*) – Cut-off frequency for lowpass filter, in Hz. Default is 10.
- **axis** (*int, optional*) – Along which axis to apply low pass filtering. Default is 0.

Returns Low pass filtered signal with the same shape as *F*.

Return type numpy.ndarray

fissa.neuropil.**separate**(*S*, *sep_method='nmf'*, *n=None*, *maxiter=10000*, *tol=0.0001*, *random_state=892*, *maxtries=10*, *W0=None*, *H0=None*, *alpha=0.1*)

For the signals in S, finds the independent signals underlying it, using ica or nmf.

Parameters

- **S** (*array_like*) – 2-d array containing mixed input signals. Each column of *S* should be a different signal, and each row an observation of the signals. For *S[i,j]*, *j* = each signal, *i* = signal content. The first column, *j = 0*, is considered the primary signal and the one for which we will try to extract a decontaminated equivalent.
- **sep_method** (*{'ica', 'nmf'}*) – Which source separation method to use, either ICA or NMF.
  - *'ica'*: Independent Component Analysis
  - *'nmf'*: Non-negative Matrix Factorization
- **n** (*int, optional*) – How many components to estimate. If *None* (default), use PCA to estimate how many components would explain at least 99% of the variance and adopt this value for *n*.
- **maxiter** (*int, optional*) – Number of maximally allowed iterations. Default is 500.
- **tol** (*float, optional*) – Error tolerance for termination. Default is 1e-5.
- **random_state** (*int, optional*) – Initial state for the random number generator. Set to None to use the numpy.random default. Default seed is 892.
- **maxtries** (*int, optional*) – Maximum number of tries before algorithm should terminate. Default is 10.
- **W0** (*array_like, optional*) – Optional starting condition for *W* in NMF algorithm. (Ignored when using the ICA method.)
- **H0** (*array_like, optional*) – Optional starting condition for *H* in NMF algorithm. (Ignored when using the ICA method.)
- **alpha** (*float, optional*) – Sparsity regularizaton weight for NMF algorithm. Set to zero to remove regularization. Default is 0.1. (Ignored when using the ICA method.)

Returns

- *numpy.ndarray* – The raw separated traces.
- *numpy.ndarray* – The separated traces matched to the primary signal, in order of matching quality (see Methods below).
- *numpy.ndarray* – Mixing matrix.
- *dict* – Metadata for the convergence result, with keys:
  - *'random_state'*: seed for ICA initiation
  - *'iterations'*: number of iterations needed for convergence
  - *'max_iterations'*: maximum number of iterations allowed
  - *'converged'*: whether the algorithm converged or not (bool)

Notes

Concept by Scott Lowe and Sander Keemink. Normalize the columns in estimated mixing matrix A so that *sum(column)=1*. This results in a relative score of how strongly each separated signal is represented in each ROI signal.

fissa.readimagejrois module

Tools for reading ImageJ files.

Based on code originally written by Luis Pedro Coelho <luis@luispedro.org>, 2012, available at https://gist.github.com/luispedro/3437255, distributed under the MIT License.

Modified

- 2014 by Jeffrey Zaremba (@jzaremba), https://github.com/losonczylab/sima
- 2015 by Scott Lowe (@scottclowe) and Sander Keemink (@swkeemink).

fissa.readimagejrois.**parse_roi_file**(*roi_source*)

Parses an individual ImageJ ROI

This implementation utilises the read_roi package, which is more robust but does only supports Python 3+ and not Python 2.7.

Parameters **roi_source** (*str or file object*) – Path to file, or file object containing a single ImageJ ROI

Returns Returns a parsed ROI object, a dictionary with either a *'polygons'* or a *'mask'* field.

Return type dict

Raises

- IOError – If there is an error reading the roi file object.
- ValueError – If unable to parse ROI.

fissa.readimagejrois.**read_imagej_roi_zip**(*filename*)

Reads an ImageJ ROI zip set and parses each ROI individually

Parameters **filename** (*str*) – Path to the ImageJ ROIs zip file

Returns **roi_list** – List of the parsed ImageJ ROIs

Return type list

fissa.roitools module

Functions used for ROI manipulation.

Authors:

- Sander W Keemink <swkeemink@scimail.eu>

fissa.roitools.**find_roi_edge**(*mask*)
Finds the outline of a mask, using the find_contour function from skimage.measure.

Parameters **mask** (*array_like*) – the mask, a binary array

Returns **outline** – Array with coordinates of pixels in the outline of the mask

Return type list of (n,2)-ndarrays

fissa.roitools.**get_mask_com**(*mask*)
Get the center of mass for a boolean mask.

Parameters **mask** (*array_like*) – A two-dimensional boolean-mask.

Returns

- *float* – Center of mass along first dimension.
- *float* – Center of mass along second dimension.

fissa.roitools.**get_npil_mask**(*mask*, *totalexpansion=4*)
Given the masks for a ROI, find the surrounding neuropil.

Parameters

- **mask** (*array_like*) – The reference ROI mask to expand the neuropil from. The array should contain only boolean values.
- **expansion** (*float, optional*) – How much larger to make the neuropil total area than mask area.

Returns A boolean numpy.ndarray mask, where the region surrounding the input is now True and the region of the input mask is False.

Return type numpy.ndarray

Notes

Our implementation is as follows:

- On even iterations (where indexing begins at zero), expand the mask in each of the 4 cardinal directions.
- On odd numbered iterations, expand the mask in each of the 4 diagonal directions.

This procedure generates a neuropil whose shape is similar to the shape of the input ROI mask.

Note: For fixed number of *iterations*, squarer input masks will have larger output neuropil masks.

---

fissa.roitools.**getmasks**(*rois*, *shpe*)
Get the masks for the specified rois.

Parameters

- **rois** (*list*) – list of roi coordinates. Each roi coordinate should be a 2d-array or equivalent list. I.e.: roi = [[0,0], [0,1], [1,1], [1,0]] or roi = np.array([[0,0], [0,1], [1,1], [1,0]]) I.e. a n by 2 array, where n is the number of coordinates. If a 2 by n array is given, this will be transposed.
- **shpe** (*array/list*) – shape of underlying image [width,height]

Returns List of masks for each roi in the rois list

Return type list of numpy.ndarray

fissa.roitools.**getmasks_npil**(*cellMask*, *nNpil=4*, *expansion=1*)
Generate neuropil masks using the get_npil_mask function.

Parameters

- **cellMask** (*array_like*) – the cell mask (boolean 2d arrays)
- **nNpil** (*int*) – number of neuropil subregions
- **expansion** (*float*) – How much larger to make neuropil subregion area than in *cellMask*

Returns Returns a list with soma + neuropil masks (boolean 2d arrays)

Return type list

fissa.roitools.**readrois**(*roiset*)
read the imagej rois in the zipfile roiset, and make sure that the third dimension (i.e. frame number) is always zero.

Parameters **roiset** (*str*) – folder to a zip file with rois

Returns Returns the rois as polygons

Return type list

fissa.roitools.**shift_2d_array**(*a*, *shift=1*, *axis=0*)
Shifts an entire array in the direction of axis by the amount shift, without refilling the array.

Parameters

- **a** (*array_like*) – Input array.
- **shift** (*int, optional*) – How much to shift array by. Default is 1.
- **axis** (*int, optional*) – The axis along which elements are shifted. Default is 0.

Returns Array with the same shape as a, but shifted appropriately.

Return type numpy.ndarray

fissa.roitools.**split_npil**(*mask*, *centre*, *num_slices*, *adaptive_num=False*)
Splits a mask into a number of approximately equal slices by area around the center of the mask.

Parameters

- **mask** (*array_like*) – Mask as a 2d boolean array.
- **centre** (*tuple*) – The center co-ordinates around which the mask will be split.
- **num_slices** (*int*) – The number of slices into which the mask will be divided.
- **adaptive_num** (*bool, optional*) – If True, the *num_slices* input is treated as the number of slices to use if the ROI is surrounded by valid pixels, and automatically reduces the number of slices if it is on the boundary of the sampled region.

Returns A list with *num_slices* many masks, each of which is a 2d boolean numpy array.

Return type list

CHAPTER 3

Contributing code

3.1 Reporting Issues

If you encounter a problem when implementing or using FISSA, we want to hear about it!

3.1.1 Gitter

To get help resolving implementation difficulties, or similar one-off problems, please ask for help on our gitter channel.

3.1.2 Reporting Bugs and Issues

If you experience a bug, please report it by opening a new issue. When reporting issues, please include code that reproduces the issue and whenever possible, an image that demonstrates the issue. The best reproductions are self-contained scripts with minimal dependencies.

Make sure you mention the following things:

- What did you do?
- What did you expect to happen?
- What actually happened?
- What versions of FISSA and Python are you using, and on which operating system?

3.1.3 Feature requests

If you have a new feature or enhancement to an existing feature you would like to see implemented, please check the list of existing issues and if you can't find it make a new issue to request it. If you do find it in the list, you can post a comment saying `+1` (or `:+1:` if you are a fan of emoticons) to indicate your support for this feature.

3.2 Documentation

We are glad to accept any sort of documentation: function docstrings, tutorials, Jupyter notebooks demonstrating implementation details, etc.

reStructuredText documents and notebooks live in the source code repository under the `docs` directory.

3.2.1 Docstrings

Documentation for classes and functions should follow the format prescribed for numpy.

A complete example of this is available here.

3.3 How to contribute

The preferred way to contribute to FISSA is to fork the main repository on GitHub.

1. Fork the project repository. Click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub server.
2. Clone this copy to your local disk:

   ```
   git clone git@github.com:YourUserName/fissa.git
   cd fissa
   ```

3. Create a branch to hold and track your changes

   ```
   git checkout -b my-feature
   ```

   and start making changes.

4. Work on this copy on your computer using Git to do the version control. When you're done editing, do:

   ```
   git add modified_files
   git commit
   ```

   to record your changes in Git, writing a commit message following the *specifications below*, then push them to GitHub with:

   ```
   git push -u origin my-feature
   ```

5. Finally, go to the web page of your fork of the FISSA repo, and click 'Pull request' to issue a pull request and send your changes to the maintainers for review. This will also send a notification email to the committers.

If any of the above seems like magic to you, then look up the Git documentation on the web.

It is recommended to check that your contribution complies with the following rules before submitting a pull request.

- All public functions and methods should have informative docstrings, with sample usage included in the doctest format when appropriate.
- All unit tests pass. Check with (from the top level source folder):

  ```
  pip install pytest
  pytest
  ```

- Code with good unit test coverage (at least 90%, ideally 100%). Check with

```
pip install pytest pytest-cov
pytest --cov=fissa --cov-config .coveragerc
```

and look at the value in the 'Cover' column for any files you have added or amended.

If the coverage value is too low, you can inspect which lines are or are not being tested by generating a html report. After opening this, you can navigate to the appropriate module and see lines which were not covered, or were only partially covered. If necessary, you can do this as follows:

```
pytest --cov=fissa --cov-config .coveragerc --cov-report html --cov-report term-
↪missing
sensible-browser ./htmlcov/index.html
```

- No pyflakes warnings. Check with:

```
pip install pyflakes
pyflakes path/to/module.py
```

- No PEP8 warnings. Check with:

```
pip install pep8
pep8 path/to/module.py
```

AutoPEP8 can help you fix some of the easier PEP8 errors.

```
pip install autopep8
autopep8 -i -a -a path/to/module.py
```

Note that using the `-i` flag will modify your existing file in-place, so be sure to save any changes made in your editor beforehand.

These tests can be collectively performed in one line with:

```
pip install -r requirements-dev.txt
pytest --flake8 --cov=fissa --cov-config .coveragerc --cov-report html --cov-report␣
↪term
```

3.3.1 Commit messages

Commit messages should be clear, precise and stand-alone. Lines should not exceed 72 characters.

It is useful to indicate the nature of your commits with a commit flag, as described in the numpy development guide.

You can use these flags at the start of your commit messages:

API: an (incompatible) API change
BLD: change related to building the package
BUG: bug fix
CI: change continuous integration build
DEP: deprecate something, or remove a deprecated object
DEV: development tool or utility
DOC: documentation; only change/add/remove docstrings, markdown or comments
ENH: enhancement; add a new feature without removing existing features
JNB: changing a jupyter notebook
MNT: maintenance commit (refactoring, typos, etc.); no functional change

REL: related to releases
REV: revert an earlier commit
RF: refactoring
STY: style fix (whitespace, PEP8)
TST: addition or modification of tests

3.4 Notes

This document was based on the contribution guidelines for sklearn, numpy and Pillow.

CHAPTER 4

Changelog

All notable changes to FISSA will be documented here.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Categories for changes are: Added, Changed, Deprecated, Removed, Fixed, Security.

4.1 Version 0.7.0

Release date: 2020-05-04. Full commit changelog.

4.1.1 Security

- **Caution:** This release knowingly exposes a new security vulnerability. In numpy 1.16, the default behaviour of numpy.load changed to stop loading files saved with pickle compression by default, due to potential security problems. However, the default behaviour of numpy.save is still to save with pickling enabled. In order to preserve our user-experience and backward compatibility with existing fissa cache files, we have changed our behaviour to allow numpy to load from pickled files. (#111)

4.1.2 Changed

- Officially drop support for Python 3.3 and 3.4. Add *python_requires* to package metadata, specifying Python 2.7 or >=3.5 is required. (#114)
- Allow tuples and other sequences to be image and roi inputs to FISSA, not just lists. (#73)
- Multiprocessing is no longer used when the number of cores is specified as 1. (#74)
- Changed default `axis` argument to internal function `fissa.roitools.shift_2d_array` from `None` to `0`. (#54)
- Documentation updates. (#112, #115, #119, #120, #121)

4.1.3 Fixed

- Allow loading from pickled numpy saved files. (#111)
- Problems reading ints correctly from ImageJ rois on Windows; fixed for Python 3 but not Python 2. This problem does not affect Unix, which was already working correctly on both Python 2 and 3. (#90)
- Reject unsupported `axis` argument to internal function `fissa.roitools.shift_2d_array`. (#54)
- Don't round number of npil segments down to 0 in `fissa.roitools.split_npil` when using `adaptive_num=True`. (#54)
- Handling float `num_slices` in `fissa.roitools.split_npil`, for when `adaptive_num=True`, which was causing problems on Python 3. (#54)

4.1.4 Added

- Test suite additions. (#54, #99)

4.2 Version 0.6.4

Release date: 2020-04-07. Full commit changelog.

This version fully supports Python 3.8, but unfortunately this information was not noted correctly in the PyPI metadata for the release.

4.2.1 Fixed

- Fix multiprocessing pool closure on Python 3.8. (#105)

4.3 Version 0.6.3

Release date: 2020-04-03. Full commit changelog.

4.3.1 Fixed

- Specify a maximum version for the panel dependency of holoviews on Python <3.6, which allows us to continue supporting Python 3.5, otherwise dependencies fail to install. (#101)
- Save deltaf to MATLAB compatible output. (#70)
- Wipe downstream data stored in the experiment object if upstream data changes, so data that is present is always consistent with each other. (#93)
- Prevent slashes in paths from doubling up if the input path has a trailing slash. (#71)
- Documentation updates. (#91, #88, #97, #89)

4.4 Version 0.6.2

Release date: 2020-03-11. Full commit changelog.

4.4.1 Fixed

- Specify a maximum version for tifffile dependency on Python <3.6, which allows us to continue supporting Python 2.7 and 3.5, which otherwise fail to import dependencies correctly. (#87)
- Documentation fixes and updates. (#64, #65, #67, #76, #77, #78, #79, #92)

4.5 Version 0.6.1

Release date: 2019-03-11. Full commit changelog.

4.5.1 Fixed

- Allow `deltaf.findBaselineF0` to run with fewer than 90 samples, by reducing the pad-length if necessary. (#62)
- Basic usage notebook wasn't supplying the correct `datahandler_custom` argument for the custom datahandler (it was using `datahandler` instead, which is incorrect; this was silently ignored previously but will now trigger an error). (#62)
- Use `ncores_preparation` for perparation step, not `ncores_separation`. (#59)
- Only use `ncores_separation` for separation step, not all cores. (#59)
- Allow both byte strings and unicode strings to be arguments of functions which require strings. Previously, byte strings were required on Python 2.7 and unicode strings on Python 3. (#60)

4.6 Version 0.6.0

Release date: 2019-02-26. Full commit changelog.

4.6.1 Added

- Python 3 compatibility. (#33)
- Documentation generation, with Sphinx, Sphinx-autodoc, and Napoleon. (#38)

4.7 Version 0.5.3

Release date: 2019-02-18. Full commit changelog.

4.7.1 Fixed

- Fix f0 detection with low sampling rates. (#27)

4.8 Version 0.5.2

Release date: 2018-03-07. Full commit changelog.

4.8.1 Changed

- The default alpha value was changed from 0.2 to 0.1. (#20)

4.9 Version 0.5.1

Release date: 2018-01-10. Full commit changelog.

4.9.1 Added

- Possibility to define custom datahandler script for other formats
- Added low memory mode option to load larger tiffs frame-by-frame (#14)
- Added option to use ICA instead of NMF (not recommended, but is a lot faster).
- Added the option for users to define a custom data and ROI loading script. (#13)

4.9.2 Fixed

- Fixed custom datahandler usage. (#14)
- Documentation fixes. (#12)

4.10 Version 0.5.0

Release date: 2017-10-05

Initial release

CHAPTER 5

Credits

5.1 Development

- Sander Keemink : Conception, Overall development
- Scott Lowe : Overall development

5.2 Supervision

- Nathalie Rochefort
- Mark van Rossum

5.3 Testing

- Janelle Pakan
- Evelyn Dylda

5.4 External Code

- ROI.py was adapted from code contained in the SIMA package under the GNU GPL v2 License.
- NIH ImageJ ROI parsing was apapted from code originally written by Luis Pedro Coelho under the MIT license.

5.5 Funding

- SK: DTC & Eurospin

- Wellcome Trust (Sir Henry Dale Fellowship), EU funding (Career Integration Grant)
- JP: EU intra-European Fellowship
- Centre for Integrative Physiology, University of Edinburgh

5.6 Citation

If you use FISSA for your research, we would be grateful if you could cite our paper on FISSA in any resulting publications:

> S. W. Keemink, S. C. Lowe, J. M. P. Pakan, E. Dylda, M. C. W. van Rossum, and N. L. Rochefort. FISSA: A neuropil decontamination toolbox for calcium imaging signals, *Scientific Reports*, **8**(1):3493, 2018. doi: 10.1038/s41598-018-21640-2.

For your convenience, we provide a copy of this citation in bibtex and RIS format.

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

f

Index

Symbols

C

E

F

G

I

L

P

R

S