Ratiometric Pharynx Analysis

Release 0.1.0

Sean Johnsen

Apr 05, 2022

CONTENTS:

1 Installation **3**
- 1.1 Basic Installation . . . 3

2 Basic Usage **5**
- 2.1 Capturing Images . . . 5
- 2.2 Organizing Files . . . 5
- 2.3 Settings File . . . 6
- 2.4 Running the Analysis . . . 8
- 2.5 Getting at the Data . . . 9

3 Developer Documentation **13**
- 3.1 Dev Environment . . . 13

4 API reference **17**
- 4.1 Experiment . . . 17
- 4.2 Pharynx IO . . . 17
- 4.3 Image Processing . . . 17
- 4.4 Profile Processing . . . 22
- 4.5 Plotting . . . 22
- 4.6 Utils . . . 22
- 4.7 Data Analysis . . . 22

5 Indices and tables **23**

Python Module Index **25**

Index **27**

This package facilitates accurate measurement of cytosolic redox state inside the pharynx of *C. elegans*, as well as the organization and analysis of that data.

CHAPTER

ONE

INSTALLATION

1.1 Basic Installation

PhaRedox is a Python package. Many operating systems ship with their own versions of Python, but these are either out of date or are dangerous to mess with (as many system tools require them). Thus, it is recommended that you install an alternate version of Python, separate from the one that ships with your computer. I suggest that you use Anaconda to manage your Python versions. Anaconda is the most popular Python environment management tool for scientists. However, if you prefer to manage your own Python environments with pyenv and pyenv-virtualenv, that will work perfectly well too. Anaconda is nice because it works the same in macOS, Windows, and Linux - which means that I only have to write one set of installation instructions!

1.1.1 Installing Anaconda

The easiest way to install Anaconda is to navigate to their download page then download and execute the appropriate file for your operating system.

1.1.2 Creating a New Environment

The fastest way to create a new environment is through the terminal (macOS/Linux) or the Anaconda Prompt (Windows, you can find this in your Start Menu). Simply execute the following command:

```
conda create --name pharedox python=3.7
```

This will create a new Python installation that is isolated from your operating system's Python installation.

1.1.3 Installing the PhaRedox Library

Now that you have a custom Python environment, we can install PhaRedox.

After activating your conda environment (*conda activate pharedox*), run:

```
pip install pharedox
```

We're almost done!

MATLAB

PhaRedox requires MATLAB for its 1D profile registration algorithm. Thus, we will need to install that MATLAB library in your new Python environment. Unfortunately, this process is difficult to automate - so it's left for you!

First, install MATLAB if you don't have it already.

> **Warning:** PhaRedox was developed using `MATLAB_R2019a`. Other versions are not guaranteed to work, though you are free to try.

Open up MATLAB. Look for `Set Path` in the `Home` tab. Click it, then in the dialog, click `Add with subfolders`, and navigate to the PhaRedox source directory and select the `matlab` folder. Finally, hit `Save`.

Next, at the MATLAB command prompt, type:

```
matlabroot
```

The system should print out a path that looks something like (on macOS):

```
'/Applications/MATLAB_R2019a.app'
```

On macOS/Linux, execute the following commands in Terminal. On Windows, execute them in the Anaconda Prompt (you can find this in the Start Menu), and replace `/` with `\`. In either case, replace `<matlabroot>` with the output of the above command.:

```
cd <matlabroot>/extern/engines/python
python setup.py install
```

1.1.4 Checking the Installation

If everything went well, you should have a working copy of PhaRedox, ready to analyze your redox experiments! To make sure, activate your conda environment, and type the following command:

```
pharedox --help
```

You should see something like:

```
Usage: pharedox [OPTIONS] COMMAND [ARGS]...

  Useful scripts for analyzing ratiometric microscopy data

Options:
  --debug / --no-debug
  --help                Show this message and exit.

Commands:
  analyze          Analyze an experiment
  create-settings  Create a settings file using the default template and...
  split-nc         Split an xarray DataArray into multiple Tiffs
```

CHAPTER

TWO

BASIC USAGE

This package makes analysis of your image stacks easy. Here, we will walk through a basic example of analyzing a fresh image stack from start to finish.

Before beginning, ensure that the package is successfully installed (see *Installation*).

2.1 Capturing Images

During image capture, be sure to keep track of the strain of each animal. You will need to know who is who during analysis. This is referred to as the `strain indexer`. The software expects the strain indexer in the following format:

strain	start_animal	end_animal
HD233	1	100
SAY47	101	200
HD233	201	300

2.2 Organizing Files

Create a folder on your computer with the following format (we will refer to this as the `experiment ID`):

```
YYYY_MM_DD-experiment_id
```

For example, let's say we've imaged HD233 and SAY47 at 4mm levamisole. An appropriate experiment ID might be:

```
2019_02_26-HD233_SAY47_4mm_lev
```

Once the folder is created, save your image stack as `experiment_id.tiff` and strain indexer as `experiment_id-indexer.csv` in that folder.

So in our example, we would save our image stack as `2019_02_26-HD233_SAY47_4mm_lev.tiff` and our strain indexer as `2019_02_26-HD233_SAY47_4mm_lev-indexer.csv`.

2.3 Settings File

The pipeline requires multiple paramters be set. These parameters are specified in the config file. To create a config file, navigate to your directory, then run the `pharedox create-settings` command, like so:

```
$ cd /path/to/experiment/directory
$ pharedox create-settings
```

This will create a template configuration file, which you can customize to your liking.

2.3.1 Parameters

Here is an overview of the parameters used in analysis

pipeline

These parameters have to do with the image analysis

strategy a string that will help you identify which parameters you chose (for example, `reg`). This will be appended to the date that you ran the experiment to create the name of the analysis directory.

channel_order The order in which the images were taken. Should be a comma-separated list like so: `TL, 470, 410, 470, 410`.

trimmed_profile_length The vector length of the trimmed profiles

untrimmed_profile_length The vector length of the untrimmed profiles (how many points to be sampled along the midline)

seg_threshold The threshold used for segmentation. If `seg_images.nc` is present in the `processed_images` directory, this is ignored.

measurement_order The order of the spline for interpolation to use when measuring the images. The order has to be in the range [0, 5].

measure_thickness The width of the midline to measure under (pixels)

reference_wavelength The wavelength to use as a reference for rotation, midline generation, etc.

image_register Whether or not to register the images (experimental). 0=no, 1=yes

channel_register Whether or not to perform 1D channel registration on the profiles after measurement. 0=no, 1=yes

population_register Whether or not to perform 1D position standardization on the profiles after measurement. 0=no, 1=yes.

trimmed_regions: a mapping from region name to region boundaries. The trimmed profile data will be averaged within these boundaries for the summary statistics. Should look like this:

```
pm3: 0.07, 0.28
pm4: 0.33, 0.45
pm5: 0.53, 0.70
pm6: 0.80, 0.86
pm7: 0.88, 0.96
```

untrimmed_regions: a mapping from region name to region boundaries. The untrimmed profile data will be averaged within these boundaries for the summary statistics. Should look like this:

```
pm3: 0.18, 0.33
pm4: 0.38, 0.46
pm5: 0.52, 0.65
pm6: 0.70, 0.75
pm7: 0.76, 0.82
```

redox

These parameters are used to map ratios to redox potentials

ratio_numerator the channel to use as the numerator in the ratio

ratio_denominator the channel to use as the denominator in the ratio

r_min the minimum ratio of the sensor (experimentally derived)

r_max the maximum ratio of the sensor (experimentally derived)

instrument_factor the "instrument factor" see SensorOverlord.

midpoint_potential the midpoint potential of the sensor

z z

temperature the temperature that the experiment was conducted at

registration

These parameters control how 1D registration works. They are ignored if all `pipeline.registration` is set to `0`.

n_deriv Which derivative to use to register the profiles

warp_n_basis the number of basis functions in the B-spline representation of the warp function

warp_order the order of the basis functions in the B-spline representation of the warp function

warp_lambda the smoothing constraint for the warp function

smooth_lambda the smoothing constraint for the smoothed profiles (which will be used to generate the warp functions)

smooth_n_breaks the number of breaks in the basis functions of the B-spline representation of the smoothed profiles (which will be used to generate the warp functions)

smooth_order the order of the basis functions of the B-spline representation of the smoothed profiles (which will be used to generate the warp functions)

rough_lambda the smoothing constraint of the B-spline representation for the "rough" profiles (which are the actual data to be registered)

rough_n_breaks the number of breaks in the B-spline representation for the "rough" profiles (which are the actual data to be registered)

rough_order the roughness penalty for the B-spline representation for the "rough" profiles (which are the actual data to be registered)

output:

These parameters control which files are saved after the pipeline finishes.

should_save_plots: True if True, useful plots will be auto-generated and saved in the analysis directory

should_save_profile_data: True if True, the profile data will be saved in the analysis directory (both as `.csv` and `.nc`).

should_save_summary_data: True if True, a summary table wherein each region has been averaged will be saved in the analysis directory.

2.4 Running the Analysis

Once all of the files are in place, running the analysis is easy.

2.4.1 Automated

If you are confident in the segmentation, you can run the analysis without loading up the GUI. To do this, simply execute the following command:

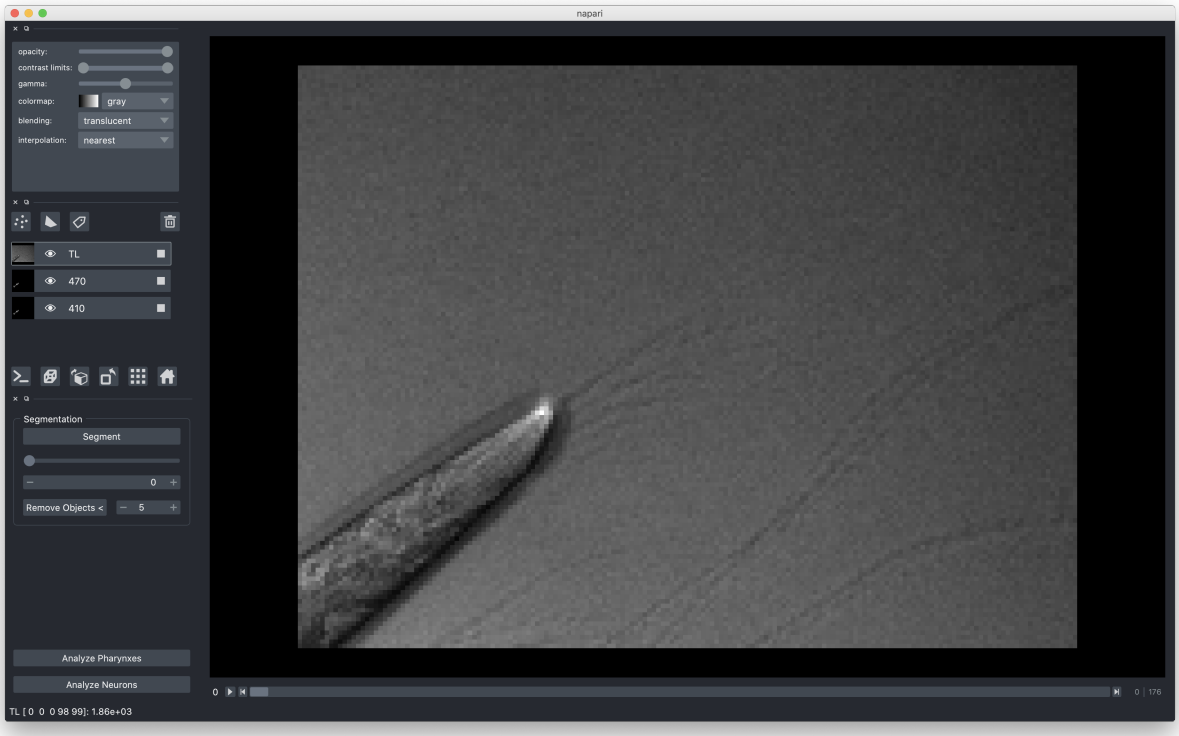
```
$ pharedox analyze --command-line "path/to/experiment directory"
```

2.4.2 GUI

The GUI (Graphical User Interface) can be helpful to make sure that your masks are correct. To launch the GUI, open a terminal, and execute the following command (make sure to include the quotation marks):

```
$ pharedox analyze "path/to/experiment directory"
```

This command will open a user interface with your images. We will use this interface to generate masks, which indicate where in each image the objects of interest are. You can hide/show each channel by clicking on the eye icon in the appropriate channel pane.



Set the threshold to a reasonable value based on your data. You can use the slider or type in the threshold box to update the threshold interactively. If your images contain small bright objects, you can use the `Remove Objects <` button to remove objects smaller than the given number.

Once you are satisfied with the masks, simply press either `Analyze Pharynxes` or `Analyze Blobs`, depending on your experiment. `Analyze Blobs` is meant for measuring neurons, the gut, or any other structure with non-stereotypical geometry.

You can monitor the status of the pipeline through the terminal with which you launched PhaRedox. If everything went well, there will be a pop-up window indicating that the pipeline has finished running. When you click `Open` you will be taken to the analysis directory containing the data from your experiment.

2.5 Getting at the Data

Each time you run an analysis, you will generate a directory within the `analyses` directory. These subdirectories are named starting with the date on which the analysis was run, and include a "strategy", which was specified in your settings file (this if for your reference, if you changed this or that setting you can come up with a name to reflect that).

After running a single analysis, the directory structure will look something like this:

```
/Users/sean/Downloads/2019_05_16_gcy8_hsf1_afd_20C
├── 2019_05_16_gcy8_hsf1_afd_20C-indexer.csv
├── 2019_05_16_gcy8_hsf1_afd_20C.tif
├── analyses
│   └── 2020-04-16_testing
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-trimmed_profile_data.csv
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-trimmed_profile_data.nc
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-trimmed_region_data.csv
```

(continues on next page)

(continued from previous page)

```
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-untrimmed_profile_data.csv
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-untrimmed_profile_data.nc
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-untrimmed_region_data.csv
│       └── figs
│           ├── 2019_05_16_gcy8_hsf1_afd_20C-movement_annotation_imgs.pdf
│           ├── 2019_05_16_gcy8_hsf1_afd_20C-ratio_images-pair=0;timepoint=0.pdf
│           └── profile_data
│               ├── trimmed_profiles
│               │   ├── avgs
│               │   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=410;pair=0;
↪timepoint=0-avgs.pdf
│               │   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=470;pair=0;
↪timepoint=0-avgs.pdf
│               │   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=e;pair=0;
↪timepoint=0-avgs.pdf
│               │   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=oxd;pair=0;
↪timepoint=0-avgs.pdf
│               │   │   └── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=r;pair=0;
↪timepoint=0-avgs.pdf
│               │   └── individual
│               │       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=410;pair=0;
↪timepoint=0-individuals.pdf
│               │       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=470;pair=0;
↪timepoint=0-individuals.pdf
│               │       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=e;pair=0;
↪timepoint=0-individuals.pdf
│               │       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=oxd;pair=0;
↪timepoint=0-individuals.pdf
│               │       └── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=r;pair=0;
↪timepoint=0-individuals.pdf
│               └── untrimmed_profiles
│                   ├── avgs
│                   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=410;pair=0;
↪timepoint=0-avgs.pdf
│                   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=470;pair=0;
↪timepoint=0-avgs.pdf
│                   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=e;pair=0;
↪timepoint=0-avgs.pdf
│                   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=oxd;pair=0;
↪timepoint=0-avgs.pdf
│                   │   └── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=r;pair=0;
↪timepoint=0-avgs.pdf
│                   └── individual
│                       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=410;pair=0;
↪timepoint=0-individuals.pdf
│                       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=470;pair=0;
↪timepoint=0-individuals.pdf
│                       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=e;pair=0;
↪timepoint=0-individuals.pdf
│                       ├── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=oxd;pair=0;
↪timepoint=0-individuals.pdf
│                       └── 2019_05_16_gcy8_hsf1_afd_20C-wavelength=r;pair=0;
↪timepoint=0-individuals.pdf
├── processed_images
│   ├── fluorescent_images
│   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=410_pair=0.tif
│   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=470_pair=0.tif
```

(continues on next page)

(continued from previous page)

```
│   │   └── 2019_05_16_gcy8_hsf1_afd_20C-wvl=TL_pair=0.tif
│   ├── rot_fl
│   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=410_pair=0.tif
│   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=470_pair=0.tif
│   │   └── 2019_05_16_gcy8_hsf1_afd_20C-wvl=TL_pair=0.tif
│   ├── rot_seg
│   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=410_pair=0.tif
│   │   ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=470_pair=0.tif
│   │   └── 2019_05_16_gcy8_hsf1_afd_20C-wvl=TL_pair=0.tif
│   └── segmented_images
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=410_pair=0.tif
│       ├── 2019_05_16_gcy8_hsf1_afd_20C-wvl=470_pair=0.tif
│       └── 2019_05_16_gcy8_hsf1_afd_20C-wvl=TL_pair=0.tif
```

CHAPTER

THREE

DEVELOPER DOCUMENTATION

3.1 Dev Environment

3.1.1 Text Editors

Of course, if you'd like to write code, you need something to write code *in*. If you don't already have a preffered code editor, I would recommend downloading Pycharm (the free version works just fine, but you can also get a free "professional" license by following the directions here. PyCharm is nice because it has built-in auto-complete while you're typing, and has a very nice debugger, which is critical for code development.

Configuring Code Formatting

This project uses the Black package to format all code automatically. This is so that all of our code "looks" the same. We'll set up your editor so that it formats the file using Black every time you save. Follow the instructions here to configure `black` to work with your text editor.

Configure Pycharm Project Interpreter

In order to do it's fancy auto-complete and other features, PyCharm needs to know which Python environment you will be using. Follow their directions here to set this up. If you are using virtual environments, use the location of that virtual environment for this step.

This section is meant to introduce new developers to the architecture / design of this system.

Data Structures

There are a few key data structures with which one needs to be familiar in order to work on this code. We deal with four main types of data. Those are:

- images
- intensity profile data in two forms:
  - raw, extracted from the images
  - functional representations of the raw data
- tabular "summary" data

Each of these types of data are naturally internally represented in different ways. We will go over each individually.

Images

Images are represented internally as numerical matrices, specifically as numpy matrices. Thinking of a single black-and-white image, each pixel is represented by a single number. The range of that number depends on the data type used to store the image internally. In our case, the raw images from our microscope come to us as 16-bit unsigned integers. This means that each pixel can take any value in the range [0, 2^16].

Still thinking of a single black-and-white image, we can access individual pixels by *indexing* into the rows and columns of that image. For example, to get the value 10 pixels down and 3 pixels across (starting from the top left), we would use the following notation:

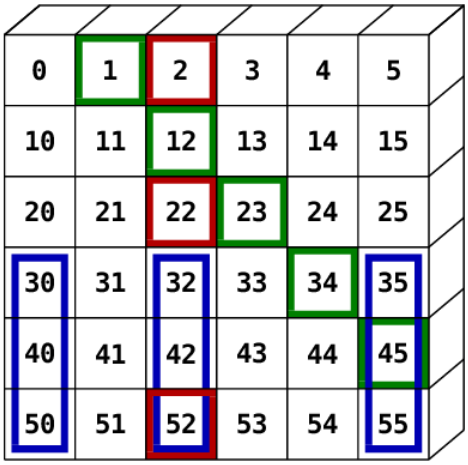
```
img[10, 3]
```

This graphic shows examples of more advanced matrix indexing.

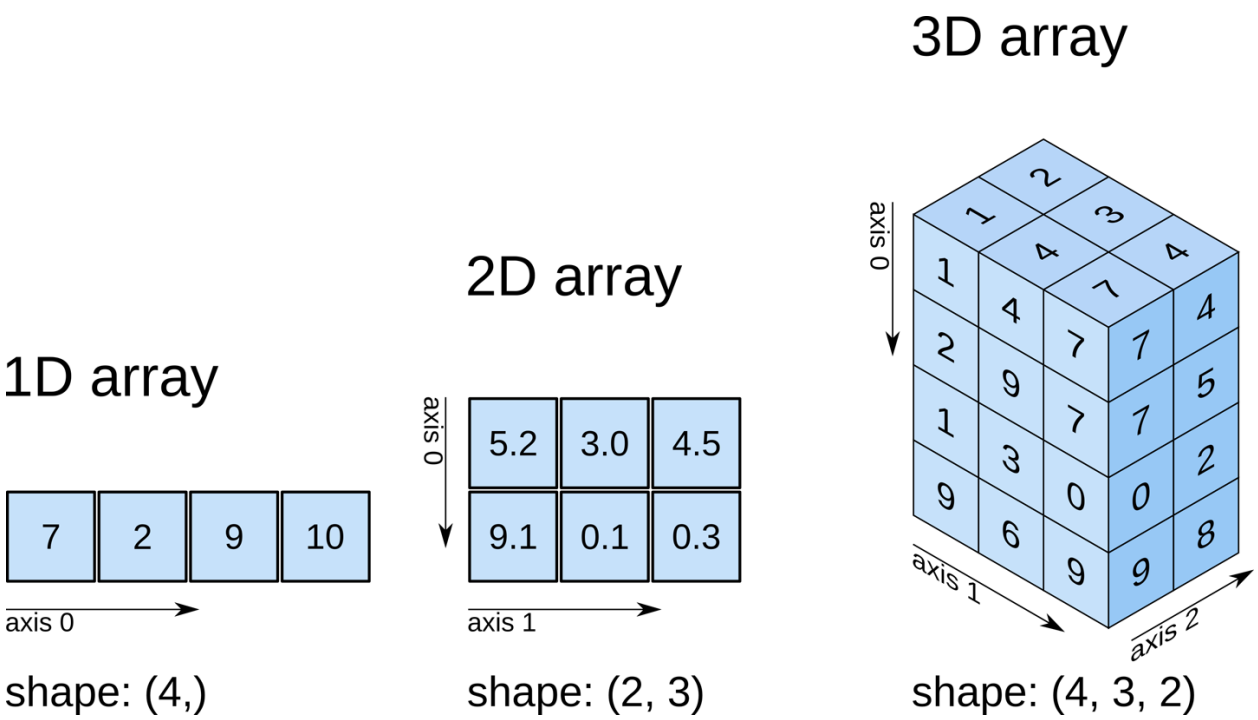
```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



Many images can be 'stacked' on top of each other, as if they were sheets of paper. Numpy handles this case as well. All that needs to be done is to add another dimension to our matrices. Now they are three-dimensional, with the first dimension indicating which sheet of paper we are dealing with, and the second and third indicating the rows and columns as before.



For an even deeper dive on indexing, see the numpy indexing documentation.

Writing New Code

Formatting

The python in this code-base is formatted via the Black package. From their docs:

```
By using Black, you agree to cede control over minutiae of hand-formatting. In
return, Black gives you speed, determinism, and freedom from pycodestyle nagging
about formatting. You will save time and mental energy for more important matters.

Black makes code review faster by producing the smallest diffs possible. Blackened
code looks the same regardless of the project you're reading. Formatting becomes
transparent after a while and you can focus on the content instead.
```

Please review their documentation to set up your IDE to auto-format your code with Black.

Documentation

All docstrings should be formatted in the Numpy docstrings format.

Adding Packages

Package management is orchestrated through Anaconda. To install a new package, use:

```
$ conda add <package>
```

To update the list of required packages, use:

```
$ conda list --explicit > conda-spec-file.txt
```

Building Documentation

This documentation is written in RST files, and built using Sphinx. All documentation should be written in `docs/source`. This documentation is auto-built and uploaded to readthedocs on push. To build the documentation as HTML on your local machine, use:

```
$ cd docs
$ make html
```

The output is then in `docs/build/html`

CHAPTER

FOUR

API REFERENCE

This page provides an auto-generated summary of the APIs used for pharynx analysis.

4.1 Experiment

4.2 Pharynx IO

This module coordinates loading data from and saving data to disk.

4.3 Image Processing

This module contains the code for processing images, including segmentation, midline calculation and measurement, translation, binary morphological operations, etc.

pharedox.image_processing.**calculate_midline**(*rot_seg_img*, *degree=4*, *pad=10*)

Calculate a the midline for a single image by fitting a polynomial to the segmented pharynx

Parameters

- **rot_seg_img** (*Union[np.ndarray, xr.DataArray]*) – The rotated masked pharynx image
- **degree** (int) – the degree of the polynomial
- **pad** (int) – the number of pixels to "pad" the domain of the midline with respect to the boundaries of the segmentation mask

Returns the estimated midline

Return type Polynomial

Notes

Right now this only works for images that this have been centered and aligned with their anterior-posterior along the horizontal.

pharedox.image_processing.**calculate_midlines**(*rot_seg_stack*, *degree=4*)
Calculate a midline for each animal in the given stack

Parameters

- **rot_seg_stack** (DataArray) – The rotated mask with which midlines should be calculated.
- **degree** (int) – The degree of the polynomial fit

Returns a DataArray containing the midline objects

Return type midlines

See also:

calculate_midline()

pharedox.image_processing.**center_and_rotate_pharynxes**(*fl_images*, *seg_images*)
Given a fluorescence stack and a pharyngeal mask stack, center and rotate each frame of both the FL and mask such that the pharynx is in the center of the image, with its anterior on the left.

Parameters

- **fl_images** (DataArray) – The fluorescence images to rotate and align
- **seg_images** (DataArray) – The segmented images to rotate and align

Returns A 2-tuple where the first item is the rotated fluorescence stack and the second is the rotated mask stack

Return type (rotated_fl_stack, rotated_seg_stack)

pharedox.image_processing.**create_normed_rgb_ratio_stack**(*r_imgs*, *seg_imgs*, *vmin=-7*, *vmax=7*, *cmap='coolwarm'*, *output_filename=None*)
Z-normalize the images (relative to the masks), then transform them into RGB with the given colormap

pharedox.image_processing.**extract_largest_binary_object**(*bin_img*)
Extracts the largest binary object from the given binary image

Parameters **bin_img** (Union[DataArray, ndarray]) – The binary image to process

Returns The binary image containing only the largest binary object from the input

Return type bin_img

pharedox.image_processing.**get_area_of_largest_object**(*mask*)
Returns the area (px) of the largest object in a binary image

Parameters **mask** (*np.ndarray*) – the binary image

Returns the area of the largest object

Return type int

pharedox.image_processing.**get_lr_bounds**(*rot_seg_stack*, *pad=0*, *ref_wvl='410'*, *ref_pair=0*)
Get the left and right boundaries of the rotated pharynxes

Parameters

- **rot_seg_stack** (DataArray) – the rotated segmented pharynxes
- **pad** (int) – the amount of padding on the left/right of the bounds
- **ref_wvl** (str) – the wavelength to use for calculating bounds
- **ref_pair** (int) – the pair to use for calculating bounds

Returns An (m, 2) array where m = number of animals, the first column is the left bound and the second column is the right bound

Return type bounds

pharedox.image_processing.**measure_under_labels**(*imgs*, *masks*, *ref_wvl='410'*, *ratio_numerator='410'*, *ratio_denominator='470'*)

Measure the intensities of each channel under the label image

pharedox.image_processing.**measure_under_midline**(*fl*, *mid*, *n_points=100*, *thickness=0.0*, *order=1*, *norm_scale=1*, *flatten=True*)

Measure the intensity profile of the given image under the given midline at the given x-coordinates.

Parameters

- **flatten** –
- **norm_scale** –
- **order** – the interpolation order
- **fl** (DataArray) – The fluorescence image to measure
- **mid** (Polynomial) – The midline under which to measure
- **n_points** (int) – The number of points to measure under
- **thickness** (float) – The thickness of the line to measure under.

Notes

Using thickness is slower, depending on the amount of thickness

On my machine (2GHz Intel Core i5), as of 12/4/19:

0-thickness: 492 µs ± 16.6 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

2-thickness: 1.99 ms ± 65.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

10-thickness: 3.89 ms ± 92.1 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

Returns **zs** – The intensity profile of the image measured under the midline at the given x-coordinates.

Return type np.ndarray

pharedox.image_processing.**measure_under_midlines**(*fl_stack*, *midlines*, *n_points=300*, *order=1*, *thickness=0*)

Measure under all midlines in stack

Parameters

- **order** –
- **fl_stack** (DataArray) – The fluorescence stack under which to measure

- **midlines** (*dict*) – A DataArray containing the midlines
- **n_points** (*int*) – the number of points to sample under the midline
- **thickness** (*float*) – the thickness of the midline to measure under

Returns **profile_data** – the intensity profiles for each image in the stack

Return type xr.DataArray

pharedox.image_processing.**normalize_images_by_wvl_pair**(*fl_imgs*, *profiles*, *percent_to_clip=2.0*)

Normalize images by subtracting mean profile then min-max rescaling to [0, 1] :type fl_imgs: DataArray :param fl_imgs: the images to normalize :type profiles: DataArray :param profiles: the intensity profiles corresponding to the images :type percent_to_clip: float :param percent_to_clip: how much to clip the profile when calculating mean/min/max, expressed as a percentage of the length of the profile

Returns the normalized images

Return type xr.DataArray

pharedox.image_processing.**normalize_images_single_wvl**(*fl_imgs*, *profiles*, *percent_to_clip=2.0*)

Normalize single wavelength image stack by subtracting the mean of the corresponding intensity profile, then min-max rescaling to [0, 1]

Parameters

- **fl_imgs** (Union[ndarray, DataArray]) – an array-like structure of shape (frame, row, col)
- **profiles** (Union[ndarray, DataArray]) – an array-like structure of shape (frame, position_along_midline)
- **percent_to_clip** (float) – how much to clip the profile when calculating mean/min/max, expressed as a percentage of the length of the profile

Returns normalized images

Return type Union[np.ndarray, xr.DataArray]

pharedox.image_processing.**rotate**(*img*, *tform*, *orientation*, *order=1*, *preserve_range=True*)

Rotate the given image with the given translation matrix and orientation angle

Parameters

- **img** (Union[ndarray, DataArray]) – the image to rotate
- **tform** – the translation matrix to apply
- **orientation** – the angle of orientation (radians)
- **order** – the order of the interpolation
- **preserve_range** – preserve the input data range

Returns the translated and rotated image

Return type rotated

pharedox.image_processing.**segment_pharynx**(*fl_img*, *target_area=450*, *area_range=100*)

Generate a mask for the given image containing a pharynx.

Parameters

- **fl_img** (*xr.DataArray*) – a fluorescent image containing a single pharynx

- **target_area** (*int, optional*) – the presumptive area (in px) of a pharynx, by default 450
- **area_range** (*int, optional*) – the acceptable range (in px) above/below the target_area, by default 100

Returns an image containing the segmented pharynx (dtype: np.uint8). Pixels of value=1 indicate the pharynx, pixels of value=0 indicate the background.

Return type xr.DataArray

pharedox.image_processing.**segment_pharynxes**(*fl_stack*, *wvl='410'*, *target_area=450*, *area_range=100*)

Segment a hyperstack of pharynxes

Parameters

- **fl_stack** (*xr.DataArray*) – the fluorescent images to segment
- **wvl** (*str, optional*) – the wavelength to segment, by default "410"
- **target_area** (*int, optional*) – the presumptive area of a pharynx, in pixels, by default 450
- **area_range** (*int, optional*) – the acceptable range of pharyngeal areas, by default 100

Returns the masks for the specified wavelength

Return type xr.DataArray

pharedox.image_processing.**shift**(*image*, *vector*)

Translate the image according to the given movement vector

Parameters

- **image** (ndarray) – the image to translate
- **vector** (ndarray) – translation parameters `(dx, dy)`

Returns **img** – the translated image

Return type np.ndarray

pharedox.image_processing.**subtract_medians**(*data*, *image_data=None*)

Subtract medians from data, optionally calculating them from a separate piece of data.

Parameters

- **data** (DataArray) – the data to subtract the median from.
- **image_data** (Optional[DataArray]) – the data to calculate the median with. Must include the dimensions *x* and *y*. If specified, all other dimensions must be identical to those in *data*. If not specified, the medians will be calculated with *data*.

Return type DataArray

pharedox.image_processing.**z_normalize_with_masks**(*imgs*, *masks*)

Perform z-normalization [0] on the entire image (relative to the content within the masks).

That is to say, we center the pixels (within the mask) such that their mean is 0, and ensure their standard deviation is ~1.

This allows us to see spatial patterns within the masked region (even if pixels outside of the masked region fall very far above or below those inside) by setting the colormap center around 0.

[0] - https://jmotif.github.io/sax-vsm_site/morea/algorithm/znorm.html

4.4 Profile Processing

This module contains the code for processing the measured profiles, including functions for transforming ratios to OxD and E, trimming profiles, and registering profiles.

4.5 Plotting

This module contains plotting code

4.6 Utils

4.7 Data Analysis

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pharedox.image_processing, 17

INDEX

C

calculate_midline() (*in module pharedox.image_processing*), 17
calculate_midlines() (*in module pharedox.image_processing*), 18
center_and_rotate_pharynxes() (*in module pharedox.image_processing*), 18
create_normed_rgb_ratio_stack() (*in module pharedox.image_processing*), 18

E

extract_largest_binary_object() (*in module pharedox.image_processing*), 18

G

get_area_of_largest_object() (*in module pharedox.image_processing*), 18
get_lr_bounds() (*in module pharedox.image_processing*), 18

M

measure_under_labels() (*in module pharedox.image_processing*), 19
measure_under_midline() (*in module pharedox.image_processing*), 19
measure_under_midlines() (*in module pharedox.image_processing*), 19
module
    pharedox.image_processing, 17

N

normalize_images_by_wvl_pair() (*in module pharedox.image_processing*), 20
normalize_images_single_wvl() (*in module pharedox.image_processing*), 20

P

pharedox.image_processing
    module, 17

R

rotate() (*in module pharedox.image_processing*), 20

S

segment_pharynx() (*in module pharedox.image_processing*), 20
segment_pharynxes() (*in module pharedox.image_processing*), 21
shift() (*in module pharedox.image_processing*), 21
subtract_medians() (*in module pharedox.image_processing*), 21

Z

z_normalize_with_masks() (*in module pharedox.image_processing*), 21