Normalization Documentation

Release 0

Jean Bilheux

Aug 18, 2017

Contents

CHAPTER 1

Installation

First you need to install the NeuNorm library

$ pip install NeuNorm

Then in your python environment, import

```
>>> import NeuNorm as neunorm
>>> from NeuNorm.normalization import Normalization
>>> from NeuNorm.roi import ROI
```

CHAPTER 2

Loading

Loading via folders

Let's pretend that our images are in the folder **/Users/me/sample/** and named

- image001.fits
- image002.fits
- image003.fits

```
>>> sample_folder = '/Users/me/sample/'
>>> o_norm = Normalization()
>>> o_norm.load(folder=sample_folder)
```

At this point all the data have been loaded in memory and can be accessed as followed

```
>>> image001 = o_norm.data['sample']['data'][0]
>>> image002 = o_norm.data['sample']['data'][1]
```

and the file names

```
>>> image003_file_name = o_norm.data['sample']['file_name'][2]
```

Let's now load the rest of our data, the OB and the DF

Our OB are in the folder **/Users/me/ob/** and named

- ob001.fits
- ob002.fits
- ob003.fits

```
>>> o_norm.load(folder='/Users/me/ob', data_type='ob')
```

again, all the data can be retrieved as followed

```
>>> ob1 = o_norm.data['ob']['data'][0]
>>> ob2_file_name = o_norm.data['ob']['file_name'][1]
```

For this library, DF are optional but for the sake of this exercise, let's load them

```
>>> o_norm.load(folder='/Users/me/df', data_type='df')
```

By default, a gamma filtering will take place when you load your data. You can manually turn off this filtering by adding the following False flag

```
>>> o_norm.load(folder='/Users/me/df', data_type='df', gamma_filter=False)
```

The gamma filtering is an algorithm that replaces all the very bright pixel counts with the average value of the 8 neighbors. What do we mean by very bright? The pixel counts that have 10% of their value above the average counts of the entire image. The threshold value can be change by doing

```
>>> o_norm.gamma_filter_threshold = 0.2
```

WARNING: #1 From this point, any operation on your data will overwrite the inital data loaded. Those data can be retrieved at any point by doing #2 The program won't let you run the same algorithm twice (normalization, df_correction, oscillation, rebin). But it's possible to overwrite this option by making a flag **force** equal to True. Use this feature at your own risk!

```
>>> data = o_norm.data['sample']['data']
>>> ob = o_norm.data['ob']['data']
```

Loading via individual file name

Let's pretend that our images are in the folder **/Users/me/sample/** and named

- image001.fits
- image002.fits
- image003.fits

```
>>> o_norm = Normalization()
>>> o_norm.load(file='/Users/me/sample/image001.fits')
>>> o_norm.load(file='/Users/me/sample/image002.fits')
>>> o_norm.load(file='/Users/me/sample/image003.fits')
```

At this point all the data have been loaded in memory and can be accessed as followed

```
>>> image001 = o_norm.data['sample']['data'][0]
>>> image002 = o_norm.data['sample']['data'][1]
```

and the file names

```
>>> image003_file_name = o_norm.data['sample']['file_name'][2]
```

Let's now load the rest of our data, the OB and the DF

Our OB are in the folder **/Users/me/ob/** and named

- ob001.fits
- ob002.fits

- ob003.fits

```
>>> o_norm.load(file='/Users/me/ob/ob001.fits', data_type='ob')
>>> o_norm.load(file='/Users/me/ob/ob002.fits', data_type='ob')
>>> o_norm.load(file='/Users/me/ob/ob003.fits', data_type='ob')
```

again, all the data can be retrieved as followed

```
>>> ob1 = o_norm.data['ob']['data'][0]
>>> ob2_file_name = o_norm.data['ob']['file_name'][1]
```

For this library, DF are optional but for the sake of this exercise, let's load them

- df001.fits
- df002.fits

```
>>> o_norm.load(file='/Users/me/df/df001.fits', data_type='df')
>>> o_norm.load(file='/Users/me/df/df002.fits', data_type='df')
```

By default, a gamma filtering will take place when you load your data. You can manually turn off this filtering by adding the following False flag

```
>>> o_norm.load(file='/Users/me/df/df002.fits', data_type='df', gamma_filter=False)
```

The gamma filtering is an algorithm that replaces all the very bright pixel counts with the average value of the 8 neighbors. What do we mean by very bright? The pixel counts that have 10% of their value above the average counts of the entire image. The threshold value can be change by doing

```
>>> o_norm.gamma_filter_threshold = 0.2
```

WARNING: #1 From this point, any operation on your data will overwrite the inital data loaded. Those data can be retrieved at any point by doing #2 The program won't let you run the same algorithm twice (normalization, df_correction, oscillation, rebin). But it's possible to overwrite this option by making a flag **force** equal to True. Use this feature at your own risk!

```
>>> data = o_norm.data['sample']['data']
>>> ob = o_norm.data['ob']['data']
```

Loading via list file names

Let's pretend that our images are in the folder **/Users/me/sample/** and named

- image001.fits
- image002.fits
- image003.fits

But from this list, we only want to load image001 and image002. It is possible to specify a list of file names to load

```
>>> o_norm = Normalization()
>>> list_files = ['/Users/me/sample/image001.fits', '/Users/me/sample/image002.fits']
>>> o_norm.load(file=list_files)
```

At this point all the data have been loaded in memory and can be accessed as followed

```
>>> image001 = o_norm.data['sample']['data'][0]
>>> image002 = o_norm.data['sample']['data'][1]
```

and the file names

```
>>> image002_file_name = o_norm.data['sample']['file_name'][1]
```

Let's now load the rest of our data, the OB and the DF

Our OB are in the folder **/Users/me/ob/** and named

- ob001.fits
- ob002.fits

```
>>> list_ob = [/Users/me/ob/ob001.fits', '/Users/me/ob/ob002.fits']
>>> o_norm.load(file=list_ob, data_type='ob')
```

again, all the data can be retrieved as followed

```
>>> ob1 = o_norm.data['ob']['data'][0]
>>> ob2_file_name = o_norm.data['ob']['file_name'][1]
```

For this library, DF are optional but for the sake of this exercise, let's load them

- df001.fits
- df002.fits

```
>>> list_df = ['/Users/me/df/df001.fits', '/Users/me/df/df002.fits']
>>> o_norm.load(file=list_df, data_type='df')
```

By default, a gamma filtering will take place when you load your data. You can manually turn off this filtering by adding the following False flag

```
>>> o_norm.load(file=list_df, data_type='df', gamma_filter=False)
```

The gamma filtering is an algorithm that replaces all the very bright pixel counts with the average value of the 8 neighbors. What do we mean by very bright? The pixel counts that have 10% of their value above the average counts of the entire image. The threshold value can be change by doing

```
>>> o_norm.gamma_filter_threshold = 0.2
```

WARNING: #1 From this point, any operation on your data will overwrite the inital data loaded. Those data can be retrieved at any point by doing #2 The program won't let you run the same algorithm twice (normalization, df_correction, oscillation, rebin). But it's possible to overwrite this option by making a flag **force** equal to True. Use this feature at your own risk!

```
>>> data = o_norm.data['sample']['data']
>>> ob = o_norm.data['ob']['data']
```

Loading via arrays

Let's pretend that our images are in the folder **/Users/me/sample/** and named

- image001.tif

- image002.tif
- image003.tif

In order to load the arrays, we first need to load ourselves the data

```
>>> data = []
>>> from PIL import Image
>>> _data1 = Image.open('/Users/me/sample/image001.tif')
>>> data.append(_data1)
>>> _data2 = Image.open('/Users/me/sample/image002.tif')
>>> data.append(_data2)
>>> _data3 = Image.open('/Users/me/sample/image003.tif')
>>> data.append(_data3)
```

Now, we can load the data

```
>>> o_norm = Normalization()
>>> o_norm.load(data=data)
```

At this point all the sample data have been loaded in memory and can be accessed as followed

```
>>> image001 = o_norm.data['sample']['data'][0]
>>> image002 = o_norm.data['sample']['data'][1]
```

and the file names

```
>>> image003_file_name = o_norm.data['sample']['file_name'][2]
```

Let's now load the rest of our data, the OB and the DF

Our OB are in the folder **/Users/me/ob/** and named

- ob001.tif
- ob002.tif
- ob003.tif

```
>>> _ob1 = Image.open('/Users/me/sample/ob001.tif')
>>> o_norm.load(data=_ob1, data_type='ob')
>>> _ob2 = Image.open('/Users/me/sample/ob002.tif')
>>> o_norm.load(data=_ob2, data_type='ob')
>>> _ob3 = Image.open('/Users/me/sample/ob003.tif')
>>> o_norm.load(data=_ob3, data_type='ob')
```

again, all the data can be retrieved as followed

```
>>> ob1 = o_norm.data['ob']['data'][0]
>>> ob2_file_name = o_norm.data['ob']['file_name'][1]
```

For this library, DF are optional but for the sake of this exercise, let's load them

- df001.tif
- df002.tif

```
>>> _df1 = Image.open('/Users/me/sample/df001.tif')
>>> o_norm.load(data=_df1, data_type='df')
>>> _df2 = Image.open('/Users/me/sample/df002.tif')
>>> o_norm.load(data=_df2, data_type='df')
```

By default, a gamma filtering will take place when you load your data. You can manually turn off this filtering by adding the following False flag

```
>>> o_norm.load(data=_df2, data_type='df', gamma_filter=False)
```

The gamma filtering is an algorithm that replaces all the very bright pixel counts with the average value of the 8 neighbors. What do we mean by very bright? The pixel counts that have 10% of their value above the average counts of the entire image. The threshold value can be change by doing

```
>>> o_norm.gamma_filter_threshold = 0.2
```

WARNING: #1 From this point, any operation on your data will overwrite the inital data loaded. Those data can be retrieved at any point by doing #2 The program won't let you run the same algorithm twice (normalization, df_correction, oscillation, rebin). But it's possible to overwrite this option by making a flag **force** equal to True. Use this feature at your own risk!

```
>>> data = o_norm.data['sample']['data']
>>> ob = o_norm.data['ob']['data']
```

CHAPTER 3

Dark Field Correction

If you loaded a set of Dark Field (DF) images, you probably want to correct all your images (sample and OB) for dark field correction

```
>>> o_norm.df_correction()
```

In case you did not loaded a set of DF, this correction will leave the images untouched

CHAPTER 4

Normalization

Normalization using ROI (optional)

If you want to specify a region of your sample to match with the OB

Let's use the following region

- x0 = 10
- y0 = 10
- x1 = 50
- y1 = 50

```
>>> my_norm_roi = ROI(x0=10, y0=10, x1=50, y1=50)
```

then the normalization can be run

```
>>> o_norm.normalization(roi=my_norm_roi)
```

Normalization without ROI (optional)

If you don't want any normalization ROI, simply run the normalization

```
>>> o_norm.normalization()
```

How to get the normalized data

Each of the data set in the sample and ob will then be normalized. If a norm_roi has been provided, the sample arrays will be divided by the average of the region defined. Same thing for the ob. Those normalized array can be retrieved this way

```
>>> sample_normalized_array = o_norm.data['sample']['data']
>>> ob_normalized_array = o_gretting.data['ob']['data']
```

CHAPTER 5

Cropping Data

You have the option to crop the data but if you do, this must be done after running the normalization. The algorithm only cropped the normalized sample and ob data

- the 4 corners of the region of interest (ROI)
- the top left corner coordinates, width and height of the ROI

let's use the first method and let's pretend the ROI is defined by

- x0 = 5
- y0 = 5
- x1 = 200
- y1 = 250

```
>>> my_crop_roi = ROI(x0=5, y0=5, x1=200, y1=250)
>>> o_norm.crop(roi=my_crop_roi)
```

CHAPTER 6

Retrieve Normalized Data

The sample/OB normalized data can be recovered this way

```
>>> normalized_data = neunorm.data['normalized']
```

You can retrieve the data using either this way

```
>>> sample = o_norm.data['sample']['data']
>>> ob = o_norm.data['ob']['data']
>>> df = o_norm.data['df']['data']
>>> norm = o_norm.data['normalization']
```

or

```
>>> sample = o_norm.get_sample_data()
>>> ob = o_norm.get_ob_data()
>>> df = o_norm.get_df_data()
>>> normalized = o_norm.get_normalized_data()
```

CHAPTER 7

Export Data

It is possible to export any of the data you worked on (sample, ob, df or normalized) either as a 'tif' or as a 'fits' file (default being 'tif')

```
>>> output_folder = '/users/my_output_folder'
>>> o_norm.export(folder=output_folder, data_type='normalized', file_type='tif')
```

or if you prefer 'fits'

```
>>> o_norm.export(folder=output_folder, data_type='normalized', file_type='fits')
```

CHAPTER 8

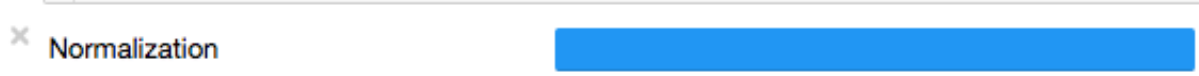
Using library from a Notebook

If you run the library from a notebook, you have the option to display a progress bar showing you the progress of the loading or normalization processes (NB: progress bar will not show up when loading one file at a time)

```
>>> sample_folder = '/Users/me/sample/'
>>> o_norm = Normalization()
>>> o_norm.load(folder=sample_folder, notebook=True)
```

or during normalization

```
>>> o_norm.normalization(notebook=True)
```



CHAPTER 9

Indices and tables

- genindex
- modindex
- search