inpystem

Release 0.1

Etienne Monier

Apr 20, 2021

CONTENTS:

1 inpystem User Guide 1

2 inpystem Developper Guide 21

3 Full inpystem API documentation 27

4 Indices and tables 63

Bibliography 65

Python Module Index 67

Index 69

CHAPTER

ONE

INPYSTEM USER GUIDE

1.1 Introduction

1.1.1 What is inpystem

inpystem is an open source Python library which provides tools to reconstruct partially sampled 2D images as multi-band images.

inpystem's core is a set of reconstruction techniques such as interpolation, regularized least-square and dictionary learning methods. It provides a user interface which simplify the use of these techniques.

inpystem is mainly at the destination of the microscopy community so that it highly depends on the good library HyperSpy.

This library was originally developed by its creator Etienne Monier to handle EELS data and develop reconstruction algorithms. This was proposed afterwards to the microscopy community as a tool.

1.1.2 About the developer

This library is developed by Etienne Monier, a French PhD student.

His research interests are in new methods and algorithms to solve challenging acquisition problems encountered in the acquisition of multi-band microscopy images. In particular, he is interested in acquiring high-SNR Electron Energy Loss Spectroscopy (EELS) images with extremely low beam energy to prevent destruction of sensitive microscopy samples. His goal is to provide to the EELS community precise algorithms to detect rapidly the presence of a chemical element inside a sample to reduce irradiation as much as possible. Such problem requires **signal processing** methods, such as **convex optimization** and **proximal splitting** methods.

1.2 Installing inpystem

1.2.1 With pip

inpystem is listed in the Python Package Index. Therefore, it can be automatically downloaded and installed with pip. You may need to install pip for the following commands to run.

Install with pip:

```
$ pip install inpystem
```

1.2.2 Install from source

When installing manually, be sure that all dependences are required. For example, do:

```
$ pip install numpy scipy matplotlib hyperspy scikit-learn scikit-image
```

Be aware that the scikit-image package versions is above 0.16.

Released version

To install from source grab a tar.gz release from *Python Package Index <https://pypi.org/>* and use the following code if Linux/Mac user:

```
$ tar -xzf inpystem.tar.gz
$ cd inpystem
$ python setup.py install
```

You can also use a Python installer, e.g.

```
$ pip install inpystem.tar.gz
```

Development version

To get the development version from our git repository you need to install git. Then just do:

```
$ git clone https://github.com/etienne-monier/inpystem
$ cd inpystem
```

Then, perform one of the following commands:

```
$ pip install -e .
$ python setup.py install
```

1.3 Getting started

In this documentation, we will assume that the reader can write some command line or jupyter python.

1.3.1 Starting inpystem in python

inpystem can be imported in python just as any python package.

```
>>> import inpystem
```

In addition to inpystem, the HyperSpy library is required to construct the inpystem objects as some arguments should be HyperSpy data.

```
>>> import hyperspy.api as hs
```

Last, note that most of the scientific python applications require libraries such as numpy and matplotlib. It is recommended to import them as well.

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

1.3.2 The base: What is a STEM acquisition here ?

A STEM acquisition here is the result of two main informations:

- **The scan pattern** which is basically the array of the visited pixels indexes,
- **The data** which are the values of the data at the sampled pixels.

Be aware that the inpystem library can handle 2D (e.g. HAADF data) as 3D data (e.g. EELS data). The data that are acquired at a spatial position (or pixel) can then be a single value (for 2D) or a spectrum (for 3D).

1.3.3 The first basic object is the scan

The basic object to understand is the *Scan* class. This is an object which stores the scan pattern. To create one, the data spatial shape and the path indexes should be given.

```
>>> shape = (3, 4)  # This is the spatial shape of the data : 3 rows, 4 columns.
>>> path = [ 1,  3,  6,  4,  7, 11,  8,  0]  # These are the sampled pixels indexes.
>>> scan = inpystem.Scan(shape, path)
>>> scan
<Scan, shape: (3, 4), ratio: 0.667>
```

Note: Note that in this documentation, the matrix indexes are denoted in the rows major order (which is the order chosen in python).

It mean that if the spatial dimensions are `(M, N)`, then the i'th index refers to the pixels which coordinates are `(i // N, i % N)`. In the above example, the first visited pixel is the second pixel of the first line.

Be careful also that **python coordinates begin at 0**, not 1.

Additionally, the *Scan* object allows you to select a fraction of the scan pattern. Let us consider a fully sampled acquisition which spatial shape is `(M, N)`, then only 10% of the pixels can be selected by setting the `ratio` argument to `0.1`. This argument can be modified after the object definition.

Thought, be careful in case the acquisition is partially sampled (or example, if only `r*M*N` pixels were acquired, with `r` below 1). In such case, if `ratio` is below `r`, then only `ratio*M*N` pixels are kept. If `ratio` is above `r`, then all `r*M*N` pixels are kept.

```
>>> scan
<Scan, shape: (3, 4), ratio: 0.667>
>>> scan.ratio = 0.8  # Here, we ask for a ratio which is higher than 0.667.
WARNING:root:Input ratio is higher than higher maximal ratio (0.667). Ratio is set to␣
↪the maximal value.
>>> scan.ratio = 0.5  # Here, the value is correct.
>>> scan
<Scan, shape: (3, 4), ratio: 0.500>

>>> scan = inpystem.Scan(shape, path, ratio=0.5)  # The ratio can be given at␣
↪initialization.
>>> scan
<Scan, shape: (3, 4), ratio: 0.500>
```

(continues on next page)

(continued from previous page)

```
>>> scan.ratio = 0.667  # But don't worry, the additional visited pixels are not lost.
>>> scan
<Scan, shape: (3, 4), ratio: 0.667>
```

In fact, the pixels that are given at initialization of `scan` are not lost when a below `ratio` is given as the currently visited index are stored in `path` attribute while the `path_0` attribute stores all pixels at initialization.

```
>>> scan = inpystem.Scan(shape, path, ratio=0.5)  # The ratio can be given at␣
↪initialization.
>>> scan
<Scan, shape: (3, 4), ratio: 0.500>
>>> scan.path
.. code-block:: python
>>> scan.path_0
array([ 1,  3,  6,  4,  7, 11,  8,  0])
>>> scan.ratio
0.5
```

Note: The Scan object data can be represented with a sampling mask $\mathbf{M}$ defined as

$$\mathbf{M}_i = \begin{cases} 1, & \text{if pixel \# } i \text{ is acquired} \\ 0, & \text{otherwise} \end{cases}$$

This representation suffer from information deficiency, but is interesting to study the acquired pixels repartition. This sampling mask which shape is the same as the spatial shape can be obtained using the *`get_mask()`* method of *`Scan`*. This one can also be plotted using the method *`plot()`* (see *Data Visualization*).

1.3.4 The second basic object is data

Well, data here are nothing else than HyperSpy data. Please refer to its documentation for more info about it.

1.3.5 The result is inpystem data

As explained previously, the inpystem data is the combination of a *`Scan`* object and an HyperSpy data. Two classes are proposed to the user:

- *`Stem2D`* for 2D data,
- *`Stem2D`* for 3D data.

Both are initialized with a scan pattern and the associated data. Though, the scan pattern is optional as the default scan pattern is raster scan (line-by-line) full sampling.

```
>>> import hyperspy.api as hs
>>> haadf_hs = hs.load('haadf_data.dm4')
>>> acquisition_1 = inpystem.Stem2D(haadf_hs)  # fully sampled HAADF image.

>>> m, n = haadf_hs.data.shape
>>> N = int(0.5*m*n)  # The number of pixels to visit.
>>> path = np.random.permutation(m*n)[:N]
>>> scan = inpystem.Scan((m, n), path)
>>> acquisition_2 = inpystem.Stem2D(haadf_hs, scan)  # partially sampled HAADF image.
```

(continues on next page)

(continued from previous page)

```
>>> eels_hs = hs.load('eels_data.dm4')
>>> acquisition_3 = inpystem.Stem3D(eels_hs)  # fully sampled EELS image.
```

1.3.6 Loading your data is faster

inpystem offers you a way to accelerate the data definition. To that end, inpystem proposes you to setup a data directory (let's say `/my/wonderful/data/dir/`) and to put inside your data so that the structure looks like this:

```
/my/wonderful/data/dir/
|
+-- MyData1
|     |
|     +-- ells_data.dm4
|     +-- haadf_data.dm4
|     +-- scan.dm4
|     +-- MyData1.conf
|
+-- MyData2
      |
      +-- ells_data_2.dm4
      +-- MyData2.conf
```

Note: The data directory is not set by default. You should use the *`set_data_path()`* function to set the path. Then, it can be read with the *`read_data_path()`*.

```
>>> inpystem.set_data_path('/my/wonderful/data/dir/')
>>> inpystem.read_data_path()
'/my/wonderful/data/dir/'
```

The data directory contains sub-directories which host:

- the data files (2D/3D data, scan pattern),
- the **configuration file** (such as `MyData1.conf` in the above tree).

The configuration file has the structure of a `.ini` file (have a look at this page for an example format) and defines the relative location of data files. This would look like this (be aware that the section names such as `2D DATA` is case sensitive while keys such as `file` are not).

```
#
# This is a demo MyData1.conf file
#

[2D DATA]
# This section defines all info about 2D data
file = haadf_data.dm4

[3D DATA]
# This section defines all info about 3D data
File = eels_data.dm4

[SCAN]
```

(continues on next page)

(continued from previous page)

```
# This section defines all info about scan pattern
FILE = scan.dm4
```

This file defines all is necessary to define the inpystem data objects. To load the corresponding data, one should use the `load_file()` function which loads the data based on the `.conf` configuration file. Alternatively, inpystem can load the `mydata.conf` data directly by using the `load_key()` with the `mydata` key (as long as `mydata.conf` is located inside the data directory). The difference between the two functions ? `load_file()` **allows you to load a file which is not in the data directory**.

In addition to the configuration file path, the user should specify which data to load with the `ndim` argument (2 for 2D data and 3 for 3D data).

```
>>> inpystem.get_data_path()
/my/wonderful/data/dir/
>>> acquisition = inpystem.load_key('MyData1')
>>> acquisition = inpystem.load_file('/my/wonderful/data/dir/MyData2.conf', ndim=2)
```

Other arguments (such as the scan pattern ratio) can be passed to the two load function. That will be seen later.

Note: From this point, the examples can be tested directly in the command line as long as the data path is set and that the inpystem example data are downloaded and placed inside the data path. See *Some example data for fast testing* for more details.

1.3.7 What about restoration ?

Well, everything was loaded and is ready for reconstruction. Lets us consider that your acquisition was partially sampled with a ratio of 0.2. So, to use any reconstruction method, use the `restore()` method of inpystem objects.

The methods to reconstruct the data include nearest neighbor interpolation, regularized least-square and dictionary learning. Let's try with an example data (inpystem has three dataset that can be loaded easily, this will be mentioned in).

```
>>> import inpystem
>>> data = inpystem.load_key('HR-sample', ndim=2, scan_ratio=0.2)  # This loads␣
↪example data.
Reading configuration file ...
Generating data ...
Creating STEM acquisition...
Correcting STEM acquisition...

>>> data
<Stem2D, title: HR-sample, dimensions: (|113, 63), sampling ratio: 0.20>
>>> reconstructed_data, info = data.restore('interpolation', parameters={'method':
↪'nearest'})
>>> reconstructed_data
<Signal2D, title: HR-sample, dimensions: (|113, 63)>  # 2D hs data.
>>> info
{'time': 0.012229681015014648}  # Execution time in sec.
```

Have a look at the reconstructed data which is an HyperSpy data. It means that the reconstructed data can analyzed with HyperSpy tools. Additional information are returned in the `info` dictionary (for the nearest neighbor method, the only information that is returned is the execution time).

1.3.8 What about `axes_manager` and `metadata` informations ?

The initialization of *Stem2D* or *Stem3D* objects need an HyperSpy image which stores information about the axes (as the `axes_manager` attribute) and other general information (as the `metadata` attribute). **These informations are transfered to the reconstructed data**.

```
>>> data.hsdata.metadata
├── General
│   ├── original_filename = spim4-2-df-manualy aligned image.dm4
│   └── title = HR-sample
└── Signal
    ├── Noise_properties
    │   └── Variance_linear_model
    │       ├── gain_factor = 1.0
    │       └── gain_offset = 0.0
    ├── binned = False
    ├── quantity = Intensity
    └── signal_type =
>>> data.hsdata.axes_manager
<Axes manager, axes: (|113, 63)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
---------------- | ------ | ------ | ------- | ------- | ------
               x |    113 |        |      -0 |       1 |
               y |     63 |        |      -0 |       1 |

>>> reconstructed_data.metadata
├── General
│   ├── original_filename = spim4-2-df-manualy aligned image.dm4
│   └── title = HR-sample
└── Signal
    ├── Noise_properties
    │   └── Variance_linear_model
    │       ├── gain_factor = 1.0
    │       └── gain_offset = 0.0
    ├── binned = False
    ├── quantity = Intensity
    └── signal_type =
>>> reconstructed_data.axes_manager
<Axes manager, axes: (|113, 63)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
---------------- | ------ | ------ | ------- | ------- | ------
               x |    113 |        |      -0 |       1 |
               y |     63 |        |      -0 |       1 |
```

1.4 Initializing the data

1.4.1 How to initialize the scan pattern

The scan pattern can be initialized using three recipes:

- initialize it with the shape and path values (see *The first basic object is the scan*),
- initialize it with a Numpy or HyperSpy file,

- initialize it as random sampling.

Recall that a path which is initialized with the data shape only is set to be a full raster (i.e. line-by-line) scan.

Recall also that all scan initialization functions allow to define a ratio argument (see *The first basic object is the scan*).

Initialize it with a file

The scan pattern can be initialized with a numpy `.npz` file which should store:

- `m` (resp. `n`) which is the data number of rows (resp. columns),
- `path` which is the `path` argument

To that end, one should use the `from_file()` method of `Scan`.

```
>>> import numpy as np
>>> m, n = 50, 100
>>> path = np.random.permutation(m*n)
>>> data_2_save = {'m': m, 'n': n, 'path': path}
>>> np.savez('my_scan.npz', **data_2_save)  # This saves the Scan numpy file

>>> inpystem.Scan.from_file('my_scan.npz', ratio=0.5) # This loads the numpy scan␣
↪file.
<Scan, shape: (50, 100), ratio: 0.500>
```

Initialize it as random sampling

The sampling scan can last be initialized with the `random()` method of `Scan`. One should just give the spatial data shape `(m, n)`. In addition to the ratio argument which can also be given, the user can give a seed to the method to have reproducible results.

```
>>> inpystem.Scan.random((50, 100))
<Scan, shape: (50, 100), ratio: 1.000>
>>> scan = inpystem.Scan.random((50, 100), ratio=0.2)
>>> scan
<Scan, shape: (50, 100), ratio: 0.200>
>>> scan.path[:5]
array([4071,  662, 4168, 3787, 4584])

>>> scan = inpystem.Scan.random((50, 100), ratio=0.2, seed=0)
>>> scan.path[:5]
array([ 398, 3833, 4836, 4572,  636])
>>> scan = inpystem.Scan.random((50, 100), ratio=0.2, seed=0)
>>> scan.path[:5]  # This shows that setting the seed makes the results reproducible.
array([ 398, 3833, 4836, 4572,  636])
```

1.4.2 Construct inpystem data manually

As explained in *The result is inpystem data*, the inpystem data is composed of a `Scan` object which defines the sampling pattern and the HyperSpy data which stores the data. Once both have been defined, the inpystem structure can be defined by hand.

```
>>> inpystem_data = inpystem.Stem2D(hsdata, scan=scan_object)
```

1.4.3 Construct inpystem data from a Numpy array

In case your image is a numpy array, one should define the HyperSpy data before creating the inpystem data.

```
>>> import numpy as np
>>> import hyperspy.api as hs
>>> shape = (50, 100, 1500)                    # This is the 3D data shape
>>> im = np.ones(shape)                        # This is our image (which is 3D this␣
↪time).
>>> scan = inpystem.Scan.random(shape[:2])     # The scan is created (be careful to␣
↪have 2-tuple shape).
>>> hsdata = hs.signals.Signal1D(im)           # Here, hs data is created from numpy␣
↪array.
>>> inpystem.Stem3D(hsdata, scan)
<Stem3D, title: , dimensions: (100, 50|1500), sampling ratio: 1.00>
```

Well, the problem here, which is the same as for numpy-based HyperSpy data, is that both `axes_manager` and `metadata` are empty. To correct that, it is hygly recommended to use a configuration file. That's the subject of next section.

1.4.4 Construct inpystem data from a configuration file

As explained in *Loading your data is faster*, inpystem can load data from a `.conf` configuration file. This is loaded by using the `load_file()` function (or the `load_key()` function if the configuration file is in the data path). To that end, a configuration file gives to inpystem all important informations.

First, the configuration file is separated in three main sections (case-sensitive, caution !):

- `DATA 2D` for 2D data,
- `DATA 3D` for 3D data,
- `SCAN` for the scan pattern.

Among these sections, only one of `DATA 2D` and `DATA 3D` sections is required (if no data is given, inpystem can not do anything . . .). And inside this section, the only key which is required is `file` which specifies the location of the data file (numpy `.npy` or .dm4 or all other file which is allowed by HyperSpy) **relative to the configuration file**. One info: contrary to sections wich are case-sensitive, keys are not.

In case no `file` key is given inside a `SCAN` section, the `load_file()` function **creates automatically a random scan object** (based on its `scan_ratio` and `scan_seed` arguments). Otherwise, a scan file (numpy or dm4/dm3) is loaded (the `scan_ratio` argument of `load_file()` can still be given).

Hence, a basic configuration file could look like this.

```
#
# This is a demo file.
# This text is not used, that's a commentary.
#

[3D DATA]
# This section defines all info about 3D data
File = eels_data.dm4

[SCAN]
# This section defines all info about scan pattern

# If the following line is commented, the scan pattern would be random.
FILE = scan.dm4
```

In the special case where the data file is a numpy `.npy` file, one could define additional information to fill the HyperSpy `axes_manager` attribute. To that end, a set of keys can be given inside the corresponding section. These keys should be like `axis_dim_info` where:

- `dim` is the axis index (0 for the `x` axis, 1 for the `y` axis and 2 in case of 3D data for the spectrum axis),
- `info` belongs to `name`, `scale`, `unit` and `offset`.

As an example, the previous section data axes_manager should look like this.

```
>>> data = inpystem.Stem3D(hsdata, scan)
Creating STEM acquisition...

>>> data.hsdata.axes_manager
<Axes manager, axes: (100, 50|1500)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
     <undefined> |    100 |      0 |       0 |       1 | <undefined>
     <undefined> |     50 |      0 |       0 |       1 | <undefined>
---------------- | ------ | ------ | ------- | ------- | ------
     <undefined> |   1500 |        |       0 |       1 | <undefined>
```

If the numpy array is save inside a directory with the following configuration file, this issue would be fixed.

```
#
# This is a demo file to define Numpy data axes_manager.
#

[3D DATA]
file = numpy_data.npy

# Infos for the axes_manager
axis_0_name = x
axis_1_name = y
axis_2_name = Energy loss

# Some more info for the energy loss axis
axis_2_offset = 4.6e+02
axis_2_scale = 0.32
axis_2_unit = eV

# No scan section, I want a random scan.
```

And the data would be loaded by simply typing this.

```
>>> inpystem.load_file('my-nice-file.conf', scan_ratio=0.5, scan_seed=0)
```

1.4.5 Some example data for fast testing

The package is delivered with some toy data for testing which are not provided inside the package itself due to the high data size. Please download it at the github project page under location `DATA/` and copy it to your data path (see *Loading your data is faster*). These data can be called afterwards with the `load_key()` function.

The three example data are called with the following keys:

- `'HR-sample'`: this is a real atomic-scale HAADF/EELS sample,
- `'HR-synth'`: this is a synthetic EELS image generated to be similar to `'HR-sample'`,

- `'LR-synth'`: this is a synthetic low-resolution EELS image.

The first data were acquired in the context of the following works [AZWT+19], [APLML+18]. Authors of these works would like to acknowledge Daniele Preziosi for the LAO-NNO thin film growth, Alexandre Gloter for the FIB lamella preparation and Xiaoyan Li for STEM experiments.

The two last data were generated to compare reconstruction methods in the context of STEM-EELS data inpainting [AMonierOberlinBrun+18]. The high-resolution works were submitted.

1.4.6 References

1.5 Correcting data

1.5.1 Introduction

Sometimes, you can face a situation where your data has outliers, i.e. data that are abnormal because of local dysfunctions. The data *were* acquired, but their value are *not* normal. In such case, you can wish to

- purely remove columns / rows / bands,
- correct particular dead pixels by replacing their value by correct ones.

For the dead pixels, their new value are set to a mean over the nearest correct sampled positions.

In the below example, the user wants to remove the red rows, the green columns and correct a dead pixel in blue.

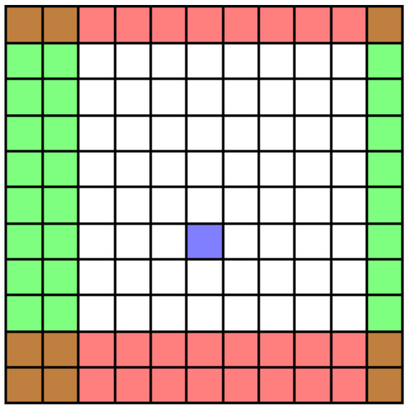

Fig. 1: Correction with a single image

1.5.2 Correcting data manually

This is realized easily by using the `correct()` method of the data object. This only requires

- slices objects that define the rows and columns to keep (and the bands in case of 3D data),
- the position of dead pixels.

Note: Let us recall what a `slice` object is. When you define a numpy array `A` of shape `(m, n)`, you can access columns and rows using e.g. `A[1:2:5, :2]`. In each direction, you specify the start, stop and step values.

For example, writing `A[1:, :]` mean you want to select all rows from row 1 to the last row `m`. This is equivalent to write `A[slice(1), :]`. In fact, a `slice` object just defines, for a direction, the start, stop and step values. Let's have some examples:

- `1:` is equivalent to `slice(1)`,
- `:10` is equivalent to `slice(None, 10)`,
- `::10` is equivalent to `slice(None, None, 10)`,
- `2:10` is equivalent to `slice(2, 10)`,
- `:` is equivalent to `slice(None)`.

For the above example, one should write the following code.

```
>>> data
<Stem3D, title: Test, dimensions: (|11, 11), sampling ratio: 0.20>
>>> data.correct(rows=slice(1, 9), columns=slice(2, 10), dpixels=[6*11+5])
Correcting STEM acquisition...

>>> data
<Stem3D, title: Test, dimensions: (|8, 8), sampling ratio: 0.20>
```

1.5.3 Correcting data with the configuration file

A lot of information can be given to inpystem through the configuration file. To pass correction info to the load functions, all you need is to define slices inside the corresponding `2D DATA` or `3D DATA` sections. For this, use the following keys: `rows`, `columns`, `bands` and `dpixels`. The slice should be written as for numpy array selection (`1:-60`). Let's illustrate is with the configuration file of the HR-sample example image.

```
[2D DATA]
file = spim4-2-df-manualy aligned image.dm4
columns = 15:-60
dpixels = [9384, 8468]

[3D DATA]
file = spim4-2_ali.dm4
columns = 15:-60
bands = 90:
dpixels = [9384, 8468]
```

Note: Contrary to the *`correct()`* method which allows slices, the configuration files require literal slices such as `1:-50`.

One other main difference is that slices objects *do not accept negative values* while the configuration files values are parsed and *accept negative values*. As a consequence, to tell inpystem you just want to keep all rows but the first one and the five last ones, you should write `slice(1:m-1)` (where `m` is the number of rows) for the *`correct()`* method while the configuration file would accept `1:-5`.

1.6 Restoration

Welcome to the main page of this documentation. The inpystem library is nothing else than a pluggin to HyperSpy to allow reconstruction.

1.6.1 Some words about pre and post-processing steps

For 2D data

2D data are centered and normalized before reconstruction. The reason is to avoid highly variable reconstruction methods parameters.

```
# data numpy array is ready to be reconstructed.
#

# Let's get data mean and standard devition.
data_mean, data_std = data.mean(), data.std()

# Let's center and normalize the data
data_ready = (data - data_mean) / data_std

#
# Reconstruction is performed
#

# Let's perform the inverse transformation
data_out = reconstructed_data * data_std + data_mean

# data_out is returned
```

For 3D data

3D data have the save pre and post-processing as for 2D data. But in addition to centering and normalization, a **thresholded principal component analysis** (PCA) is performed to reduce the data dimension along the bands axis and to ensure the low rank assumption. This one states that most multi-band data result in the mixing of `R` basic data with `R` small compared to the data dimension).

The default behavior is to perform PCA with an automatically estimated threshold (which can be really over-estimated in case of data starvation situations, i.e. if you have almost as many samples as the data size). Though, the user can set both parameters and choose if PCA should be performed and which value the threshold should have.

To sum up, the steps are:

- perform thresholded PCA if required,
- center and normalize the data,
- perform reconstruction,
- re-set the data mean and standard deviation values,
- perform inverse PCA.

1.6.2 How to reconstruct my data

To reconstruct the data, the user should use the *restore()* method. Both *Stem2D* and *Stem3D* classes need the following arguments to restore their data:

- `method` which is the method name (default is `'interpolation'`),
- `parameters` which should be a dictionary with input parameters,
- `verbose` which allows the method to display information on the console (default is `True`).

In addition to these common parameters, the *Stem3D* class has the foolowing inputs:

- `PCA_transform` which controlls the PCA execution (defualt is True for PCA execution),
- `PCA_th` which states the PCA threshold.

A common reconstruction task will then look like this.

```
>>> data = inpystem.load_example('HR-sample', ndim=2, scan_ratio=0.2)
Reading configuration file ...
Generating data ...
Creating STEM acquisition...
Correcting STEM acquisition...

>>> rec, info = data.restore()
Restoring the 2D STEM acquisition...
-- Interpolation reconstruction algorithm --
Done in 0.01s.
---
>>> rec
<Signal2D, title: HR-sample, dimensions: (|113, 63)>
>>> info
{'time': 0.011758089065551758}
```

1.6.3 The reconstruction methods available

All you need to know for each method is:

- what the method do (of course you need to know a little about it),
- his nickname to give to *restore()*,
- his parameters,
- what informations are returned.

Restoration cheet sheet

Additional info in case `PCA_transform` is `True` is `PCA_info` which stores the following keys:

- `H`: the truncated PCA basis,
- `PCA_th`: the PCA threshold,
- `Ym`: the data mean.

Interpolation

The interpolation method calls linear, cubic or nearest neighbor interpolation.

The method to give to the *`restore()`* method is `interpolation`. The associated function is resp. *`interpolate()`*.

The input parameters are:

- `method`: (optional, str) The interpolation method (among `nearest`, `linear` and `cubic`). Default is nearest neighbor.

The output dictionary stores the following informations:

- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

L1

This regularized least-square method solves the following optimization problem:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}\in\mathbb{R}^{m\times n}} \frac{1}{2}||(\mathbf{x}-\mathbf{y})\cdot\Phi||_F^2 + \lambda||\mathbf{x}\Psi||_1$$

where $\mathbf{y}$ are the corrupted data, Φ is a subsampling operator and Ψ is a 2D DCT operator.

The method to give to the *`restore()`* method is `L1`. The associated function is resp. *`L1_LS()`*.

The input parameters are:

- `Lambda`: (float) The regularization parameter,
- `init`: (optional, numpy array) An initial point for the gradient descent algorithm which should have the same shape as the input data,
- `Nit`: (optional, int) The number of iterations.

The output dictionary stores the following informations:

- `E`: The evolution of the functional value,
- `Gamma`: The set of all pixel positions which coefficient in the DCT basis is non-zero,
- `nnz-ratio`: The ratio of non-zero coefficients over the number of DCT coefficients,
- `time`: the execution time (in sec.).

Smoothed SubSpace

The 3S algorithm denoise or reconstructs a multi-band image possibly spatially sub-sampled in the case of spatially smooth images. It is well adapted to intermediate scale images.

This algorithm performs a PCA pre-processing operation to estimate:

- the data subspace basis $\mathbf{H}$,
- the subspace dimension R,
- the associated eigenvalues in decreasing order $\mathbf{d}$,
- the noise level $\hat{\sigma}$.

After this estimation step, the algorithm solves the folowing regularization problem in the PCA space:

$$\begin{aligned}\hat{\mathbf{S}} &= \underset{\mathbf{S}\in\mathbb{R}^{m\times n\times R}}{\arg\min} \frac{1}{2R}\left\|\mathbf{S}\mathbf{D}\right\|_\mathrm{F}^2 + \frac{\lambda}{2}\sum_{m=1}^{R} w_m |\mathbf{S}_{m,:}|_2^2 \\ \text{s.t.} &\quad \frac{1}{R}|\mathbf{H}_{1:R}^T\mathbf{Y}_{\mathcal{I}(n)} - \mathbf{S}_{\mathcal{I}(n)}|_2^2 \leq \alpha\hat{\sigma}^2,\ \forall n \in \{1, \dots,\ m*n\}\end{aligned}$$

where $\mathbf{Y}$ are the corrupted data, $\mathbf{D}$ is a spatial finite difference operator and $\mathcal{I}$ is the set of all sampled pixels. The coefficient α is a coefficient which scales the power of the data fidelity term.

For more details, see [BMonierOberlinBrun+18].

The method to give to the *restore()* method is `3S`. The associated function is resp. *SSS()*.

The input parameters are:

- `Lambda`: (float) The regularization parameter,
- `scale`: (optional, float) The spectr
- `init`: (optional, numpy array) An initial point for the gradient descent algorithm which should have the same shape as the input data,
- `Nit`: (optional, int) The number of iterations.

The output dictionary stores the following informations:

- `E`: The evolution of the functional value,
- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

Smoothed Nuclear Norm

The SNN algorithm denoise or reconstructs a multi-band image possibly spatially sub-sampled in the case of spatially smooth images. It is well adapted to intermediate scale images.

This algorithm solves the folowing optimization problem:

$$\hat{\mathbf{X}} = \underset{\mathbf{X}\in\mathbb{R}^{m\times n\times B}}{\arg\min} \frac{1}{2}||\mathbf{Y}_\mathcal{I} - \mathbf{X}_\mathcal{I}||_\mathrm{F}^2 + \frac{\lambda}{2}\left\|\mathbf{X}\mathbf{D}\right\|_\mathrm{F}^2 + \mu||\mathbf{X}||_*$$

where $\mathbf{Y}$ are the corrupted data, $\mathbf{D}$ is a spatial finite difference operator and $\mathcal{I}$ is the set of all sampled pixels.

For more details, see [BMonierOberlinBrun+18].

The method to give to the *restore()* method is `SNN`. The associated function is resp. *SNN()*.

The input parameters are:

- `Lambda`: (float) The λ regularization parameter,
- `Mu`: (float) The μ regularization parameter,
- `init`: (optional, numpy array) An initial point for the gradient descent algorithm which should have the same shape as the input data,
- `Nit`: (optional, int) The number of iterations.

The output dictionary stores the following informations:

- `E`: The evolution of the functional value,
- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

Cosine Least Square

The CLS algorithm denoises or reconstructs a multi-band image possibly spatially sub-sampled in the case of spatially sparse content in the DCT basis. It is well adapted to periodic data.

This algorithm solves the folowing optimization problem:

$$\hat{\mathbf{X}} = \underset{\mathbf{X}\in\mathbb{R}^{m\times n\times B}}{\arg\min} \frac{1}{2}||\mathbf{Y}_\mathcal{I} - \mathbf{X}_\mathcal{I}||_\mathrm{F}^2 + \lambda||\mathbf{X}\Psi||_{2,1}$$

where $\mathbf{Y}$ are the corrupted data, $\mathbf{D}$ is a spatial finite difference operator and $\mathcal{I}$ is the set of all sampled pixels.

The method to give to the *`restore()`* method is `CLS`. The associated function is resp. *`CLS()`*.

The input parameters are:

- `Lambda`: (float) The λ regularization parameter,
- `init`: (optional, numpy array) An initial point for the gradient descent algorithm which should have the same shape as the input data,
- `Nit`: (optional, int) The number of iterations.

The output dictionary stores the following informations:

- `E`: The evolution of the functional value,
- `Gamma`: The set of all pixel positions which coefficient in the DCT basis is non-zero,
- `nnz-ratio`: The ratio of non-zero coefficients over the number of DCT coefficients,
- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

Post-Lasso CLS algorithm

This algorithms consists in applying CLS to restore the data and determine the data support in DCT basis. A post-least square optimization is performed to reduce the coefficients bias.

The method to give to the *`restore()`* method is `Post_LS_CLS`. The associated function is resp. *`Post_LS_CLS()`*.

The input parameters are:

- `Lambda`: (float) The λ regularization parameter,
- `init`: (optional, numpy array) An initial point for the gradient descent algorithm which should have the same shape as the input data,
- `Nit`: (optional, int) The number of iterations.

The output dictionary stores the following informations:

- `E_CLS`: The evolution of the functional value for the CLS optimization step,
- `E_post_ls`: The evolution of the functional value for the post-LS optimization step,

- `Gamma`: The set of all pixel positions which coefficient in the DCT basis is non-zero,
- `nnz-ratio`: The ratio of non-zero coefficients over the number of DCT coefficients,
- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

ITKrMM and wKSVD

Weighted K-SVD (see [BMairalEladSapiro08]) and Iterative Thresholding and K residual Means for Masked data (see [BNS18]) methods.

The wKSVD and ITKrMM algorithms share a lots of their code so that their input and output are the same. Though, two implementations exist to run these algorithms: one with python (`ITKrMM` and `wKSVD` methods) and one with maltab (`ITKrMM_matlab` and `wKSVD_matlab` methods). The original Matlab codes are broadcasted by Karin Schnass. They were translated afterwards into python. Nothing distinguish them but for wKSVD where matlab is faster. The only problem is that you should have the `matlab` command in your system path.

The methods to give to the *`restore()`* method are `ITKrMM`, `wKSVD`, `ITKrMM_matlab` or `wKSVD_matlab`. The associated functions are resp. *`ITKrMM()`*, *`wKSVD()`*, `ITKrMM_matlab()` and `wKSVD_matlab()`.

The input parameters are:

- `Patchsize`: (optional, int) The patch width,
- `K`: (optional, int) The dictionary size (incl. low-rank component),
- `L`: (optional, int) The number of low-rank components to estimate,
- `S`: (optional, int) The sparsity level,
- `Nit`: (optional, int) The number of iterations for the dictionary estimation.
- `Nit_lr`: (optional, int) The number of iterations for the low-rank estimation.

The output dictionary stores the following informations:

- `dico`: The dictionary,
- `E`: The evolution of the error,
- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

BPFA

Beta Process Factor Analysis algorithm (see [BXZC+12]).

As for wKSVD and ITKrMM, BPFA is based on a Matlab code from Zhengming Xing (these codes were broadcasted without any license). The python code just calls it, so matlab should be in the path system so that the `matlab` command could be called from the command line.

The method to give to the *`restore()`* method is `BPFA_matlab`. The associated function is resp. *`BPFA_matlab()`*.

The input parameters are:

- `Patchsize`: (optional, int) The patch width,
- `K`: (optional, int) The dictionary size,
- `step`: (optional, int) That's the pixel space between two consecutive patches (if 1, full overlap),

- `Nit`: (optional, int) The number of iterations for the dictionary estimation.

The output dictionary stores the following informations:

- `dico`: The dictionary,
- `time`: the execution time (in sec.),
- `PCA_info`: in case of 3D data with PCA pre-processing, it stores info about PCA.

1.6.4 That's all folks !

This was the main content of the documentation. Congrats, you understood 90% of this library :)

1.6.5 References

1.7 Data Visualization

1.7.1 Visualizing the data

As for the HyperSpy data, the data objects `Stem2D` and `Stem3D` include a `plot()` method to display the data.

In addition to this basic function, the `Stem3D` class implements four other functions:

- `plot_sum()` which displays the sum of the 3D data along the last axis,
- `plot_as2D()` which displays the data as 2D data (the navigation direction is the "channel" axis while the data are spatial),
- `plot_as1D()` which displays the data as 1D data (the navigation directions are the spatial axes while the data is a spectrum, this is the behavior of the default `plot()` function),
- `plot_roi()` which considers the data as 1D and enable the user to mean the data over a spatial region-of-interest.

1.7.2 Visualizing the scan sampling mask

As explained in *The first basic object is the scan*, a Scan object can be represented with its sampling mask $\mathbf{M}$ defined as

$$\mathbf{M}_i = \begin{cases} 1, & \text{if pixel \# } i \text{ is acquired} \\ 0, & \text{otherwise} \end{cases}$$

This representation suffer from information deficiency, but is interesting to study the acquired pixels repartition. This sampling mask which shape is the same as the spatial shape can be obtained using the `get_mask()` method of `Scan`. This one can also be plotted using the method `plot()`.

CHAPTER

TWO

INPYSTEM DEVELOPPER GUIDE

Well, for now, this small library is developed by myself. I don't know yet how much this could be usefull. However, if someone is interested to help me, please let me know at *etienne[dot]monier[at]enseeiht[dot]fr*.

This section just give some informations about the children classes of `Stem2D` and `Stem3D` that helps me to develop algorithms.

2.1 Dev data objects

Basically, a signal processing algorithm development aims at testing some method on synthetic (i.e. noise-free) data that has been manually degraded. As the truth image is known, metrics can be computed to test and compare the method.

To that end, two data objects `Dev2D` and `Dev3D` have been developed. These are children classes from `Stem2D` and `Stem3D` (themselves are children of the basic data structure `AbstractStem`).

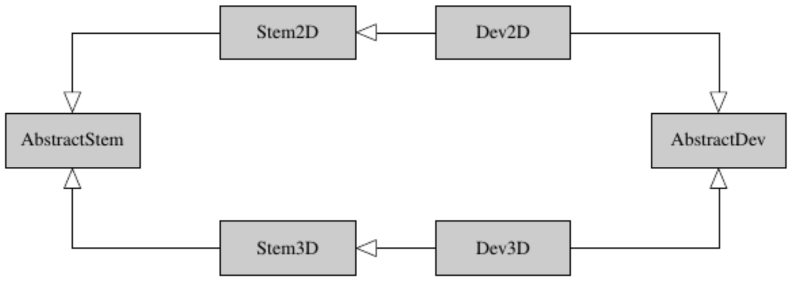

Fig. 1: A simple UML diagram.

These objects basically work as non-dev data objects:

- they are initialized with data and a Scan object,
- loading them can be done with the same functions as for non-dev data,
- they can be corrected with a configuration file,
- they can be reconstructed using the basic `restore()` of `AbstractStem`,
- they can be plotted with the same tools.

Their difference is that:

- noise can be added,

- the results are reproducible,
- new noise can be drawn whenever you want,
- the *Dev3D* allows you to consider the PCA-transformed data as the base data (this is useful for developing 3D restoration algorithm).

2.1.1 Initialize Dev data

Manually

The Dev data objects could be initialized manually with the same arguments as for *Stem2D* and *Stem3D* classes, i.e.:

- hsdata, scan and verbose for *Stem2D*,
- hsdata, scan, PCA_transform, PCA_th and verbose for *Stem3D*,

In addition to these arguments, a required input is key which is a small descriptive keyword to help referencing. Other optional inputs are:

- modif_file: an configuration file which is sent to correction function,
- sigma: the desired noise standard deviation in case additional noise is desired,
- seed: the noise seed to have reproducible data,
- normalized: if set to True, the data are normalized at initialization.

```
>>> stem2d_data = inpystem.load_key('HR-sample', 2)
Reading configuration file ...
Generating data ...
Creating STEM acquisition...
Correcting STEM acquisition...
>>> scan_shape = stem2d_data.scan.shape
>>> scan = inpystem.Scan.random(shape=scan_shape, ratio=0.5)
>>> dev_data = inpystem.Dev2D('my-dev-data', hsdata=stem2d_data.hsdata, scan=scan,␣
↪sigma=0.5, seed=0)
Creating STEM acquisition...
>>> dev_data
<Dev2D, title: HR-sample, dimensions: (|113, 63), sampling ratio: 0.50>
```

With load functions

The *load_file()* and *load_key()* functions also enable to load development data. To that end, the user just has to use the dev input which is a dictionary. This dictionary should store the desired inputs:

- for 2D data: modif_file, sigma, seed and normalized
- for 2D data: PCA_transform, PCA_th, modif_file, sigma, seed and normalized

```
>>> dev = {'sigma': 0.5, 'seed': 0}
>>> inpystem.load_key('HR-sample', 2, dev=dev, scan_ratio=0.5, scan_seed=1)
Reading configuration file ...
Generating data ...
Creating STEM acquisition...
Correcting STEM acquisition...
<Dev2D, title: HR-sample, dimensions: (|113, 63), sampling ratio: 0.50>
```

Note that in case the development data is loaded, the key would be the name of the `.conf` file (e.g. for `my-conf-file.conf`, the key would be `my-conf-file`).

2.1.2 Some words about data storage

Contrary to Stem2D and Stem3D objects, development objects work with numpy techniques (to generate the noise, add it, perform PCA) so that this is the central data to be stored in Dev objects.

More precisely, the data are stored twice or three times under the attributes:

- `data` which stores the noise-free data,
- `ndata` which stores the noisy data (in case `sigma` is None and no noise-corruption procedure was applied, this attribute is None),
- `hsdata` which stores the data as an HyperSpy data.

The last attribute only exist to send the data into restoration procedures. To display noisy or noise-free data, prefer the two first attributes.

2.1.3 Reproducibility and noise

Inside the data creation procedure, the random effects can come from the scan generation (in case of random initialization) or from the noise generation. For both situations, the seed can be set to get reproducible results. Indeed, the scan seed can be set in the load functions with its `scan_seed` parameter or directly calling the *`random()`* with its `seed` parameter. The noise seed can be set itself with the `seed` attribute.

Caution: The seed values are set for the **startup** procedures. When this is set for the *`Dev2D`* class, this seed is set just before drawing the noise matrix **for the first time**. If the user wants to draw another noise matrix, the seed will not be the same any more. This is the same for the random scans.

To generate a new noise matrix, just use the `set_ndata()`. To generate a new random scan, just re-run the *`random()`* method without the `seed` argument.

2.1.4 How PCA works for `Dev3D`

At the *`Stem3D`* initialization, the 3D data are fully stored as an HyperSpy data. When the user wants to reconstruct the data, the usual `PCA_transform` and `PCA_th` arguments can given. This is passed into the reconstruction algorithms which perform PCA as pre and post-processing steps.

In the case of the *`Dev3D`* class, these parameters are given at initialization. If `PCA_transform` is False, then the full data is stored into the `data` attribute. The user could choose to perform PCA by giving `PCA_transform` into the *`restore()`* as an argument.

```
>>> dev = {'sigma': 0.5, 'PCA_transform': False}
>>> data = inpystem.load_key('HR-sample', 3, dev=dev)
Reading configuration file ...
Generating data ...
Creating STEM acquisition...
Correcting STEM acquisition...

>>> data.data.shape
(63, 115, 1510)
```

(continues on next page)

(continued from previous page)

```
>>> outdata, info = data.restore()
Restoring the 3D STEM acquisition...
-- Interpolation reconstruction algorithm --
- PCA transformation -
Dimension reduced from 1510 to 4.
Estimated sigma^2 is 2.76e-01.
Done in 1.27s.
-
Done in 0.05s.
---

>>> info['PCA_info']['H'].shape  # The PCA basis used for restoration
(1510, 4)
```

In the case of True `PCA_transform` at initialization, a PCA procedure is executed at initialization and the `data` (and possibly `ndata`) data are reduced in the last axis direction. Additional information is stored in the `PCA_info` attribute. In such case, the user should use the `restore()` method without giving the `PCA_transform` argument.

```
>>> dev = {'sigma': 0.5, 'PCA_transform': True}
>>> data = inpystem.load_key('HR-sample', 3, dev=dev)
Reading configuration file ...
Generating data ...
Creating STEM acquisition...
Correcting STEM acquisition...
- PCA transformation -
Dimension reduced from 1510 to 290.
Estimated sigma^2 is 1.43e+03.
Done in 1.27s.
-

>>> outdata, info = data.restore()
Restoring the 3D STEM acquisition...
-- Interpolation reconstruction algorithm --
Done in 11.12s.
---

>>> 'PCA_info' in info
False
```

Note: The default behavior for the `restore()` `PCA_transform` argument is to take the logical not of the `PCA_transform` argument given at initialization. If the `Dev3D` class has been initialized without PCA, then a PCA is applied by default before restoration. If PCA has been required at initialization, then no PCA will be applied at restoration.

Yet, the user can explicitly ask for additional PCA (which is stupid, I agree) or for no PCA at all. Let's explain it clearly: **if you don't want PCA, say it at initialization and at restoration**.

Caution: The two examples above show something important as the estimated PCA threshold is 3 in the case where `PCA_transform` is not given at initialization and 290 otherwise. This means that both orders do not have the same effects.

If `PCA_transform` is True at initialization, the PCA transformation is performed **before** adding noise so that the signal is clear enough to have a high threshold. Besides, the noise is added **to the data in PCA space**.

If `PCA_transform` is False at initialization but True at restoration, the noise is added to the full dimension data. Besides, the PCA is applied to *noised* data so that few principle components get more powerful than noise and the threshold drops.

To handle easily direct and inverse PCA transformations, two methods are given: `direct_transform()` and `inverse_transform()`. They both allow the user to **perform the same PCA direct/inverse transformation as for the :class:'~.dev.Dev3D' initialization method**. These methods accept HyperSpy as numpy data.

These methods incorporate also normalization procedure inside. This means that the `direct_transform()` method performs also centering and normalization whereas the `inverse_transform()` inject the standard deviation and the mean back.

```
>>> import inpystem

# Case with non-PCA-initialized object
>>> dev = {'sigma': 0.5, 'PCA_transform': False, 'normalize': False}
>>> data = inpystem.load_key('HR-sample', 3, dev=dev)
>>> direct_data = data.direct_transform(data.data)  # Performing direct␣
↪transformation to data
>>> inverse_data = data.inverse_transform(data.data)  # Performing inverse␣
↪transformation to data
>>> import numpy.testing as npt  # This is to check arrays are equal
>>> npt.assert_allclose(data.data, direct_data)  # Equal
>>> npt.assert_allclose(data.data, inverse_data)  # Equal

>>> dev = {'sigma': 0.5, 'PCA_transform': False}  # Non-normalized here
>>> data = inpystem.load_key('HR-sample', 3, dev=dev)
>>> direct_data = data.direct_transform(data.data)
>>> inverse_data = data.inverse_transform(direct_data)
>>> npt.assert_allclose(data.data, direct_data)  # Error because of normalization
>>> npt.assert_allclose(data.data, inverse_data)  # Equal: direct, then inverse is␣
↪identity :)

# Case with PCA-initialized object
>>> dev = {'sigma': 0.5, 'PCA_transform': True}
>>> data = inpystem.load_key('HR-sample', 3, dev=dev)
>>> inverse_data = data.inverse_transform(data.data)
>>> direct_data = data.direct_transform(inverse_data)
>>> npt.assert_allclose(data.data, direct_data)  # Equal: direct, then inverse is␣
↪still identity :)

>>> data.data.shape  # PCA shape
(63, 115, 290)
>>> inverse_data.shape  # True image shape
(63, 115, 1510)
>>> direct_data.shape  # PCA shape
(63, 115, 290)
```

2.1.5 One word about visualization

The development data visualization works as for non-dev classes. Yet, development data visualization methods accept an additional argument which is `noised`. This optional argument that is False by default sets which data should be displayed (noise-free data by default or noisy data).

2.2 List of todos:

Todo: Maybe enable PCA_th in config file for 3D data.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/inpystem/checkouts/latest/inpystem/dataset.py:docstr
of inpystem.dataset.load_file, line 44.)

CHAPTER

THREE

FULL INPYSTEM API DOCUMENTATION

inpystem module.

Description here.

3.1 signals module

This package defines all sort of classes to handle data for inpystem.

3.1.1 The Scan class

class `inpystem.signals.`**`Scan`**(*shape*, *path*, *ratio=None*)

Scan pattern class.

This class stores the data spatial shape and two copies of the scan pattern. One of these copies is the initial scan pattern which is given to the class. At the same time, a `ratio` argument can be given to keep only a portion of the available samples. See Notes for more details.

Variables

- **`shape`** (*`2-length tuple`*) – The spatial shape (m, n) where m is the number of rows and n is the number of columns.
- **`path`** (*`numpy array`*) – The sampling path to be used in the study.
- **`path_0`** (*`numpy array`*) – The initial sampling path to be kept in case the ratio is changed.
- ***`ratio`*** (*`float`*) – The current `ratio` value such that `` path`has size :code:`ratio*m*n ``. Changing this attribute automaticaly updates `path`.

Note: Consider only `r*m*n` pixels hve been sampled, then the `path_0` attribute has shape (r*m*n,) and its elements lay between 0 and m*n-1.

Meanwhile, if the user wants to consider only `ratio` percent of the samples, the `ratio` argument should be given. The `path` attribute would then have shape (ratio*m*n,). In such case, `path_0[:ratio*m*n]` will be equal to `path`. Be aware that `ratio` should be lower than `r`.

Each element of these arrays is the pixel index in row major order. To recover the row and column index array, type the following commands.

::code: i = path // n j = path % n

`__init__`(*shape*, *path*, *ratio=None*)
Scan pattern constructor.

Parameters

- **`shape`** (*`(m, n) tuple`*) – The spatial shape where m is the number of rows and n is the number of columns.
- **`path`** (*`tuple, numpy array`*) – The sampling path. See class Notes for more detail.
- **`ratio`** (*`optional, float`*) – The ratio of sampled pixels. This should lay between 0 (excl.) and 1. Default is None for full sampling.

`property ratio`
Ratio getter.

Returns The ratio property value.

Return type float

`classmethod from_file`(*data_file*, *ratio=None*)
Creates a scan pattern object from a data file (such as .dm3, .dm4 or npz).

In the case of a .npz file, this one should contain the :code:'m', :code:'n' and :code:'path' variables which are resp. the number of rows and columns and the path array.

Concerning the .dm3/.dm4 files, the data storage is specific to the LPS Lab (Orsay, France) implementation.

An aditional argument :code:'ratio' allows you to select only a given ratio of the sampled pixels. This should lay between 0 (excl.) and 1.

Parameters

- **`data_file`** (*`str`*) – The data file path.
- **`ratio`** (*`optional, float`*) – The ratio of sampled pixels. This should lay between 0 (excl.) and 1. Default is None for full sampling.

Returns The scan pattern.

Return type Scan object

`classmethod random`(*shape*, *ratio=None*, *seed=None*)
Creates a random scan pattern object.

Parameters

- **`shape`** (*`(m, n) tuple`*) – The data spatial shape.
- **`ratio`** (*`optional, float`*) – The ratio of sampled pixels. It should lay between 0 (excluded) and 1. Default is None for full sampling.
- **`seed`** (*`optional, int`*) – Seed for random sampling. Default is None for random seed.

Returns The scan pattern.

Return type Scan object

`get_mask`()
Returns the sampling mask.

The sampling mask is boolean and True is for sampled pixels.

Returns **mask** – The sampling mask.

Return type (m, n) numpy array

plot()

Plots the sampling mask.

White (resp. black) pixels are sampled (resp. non-sampled).

3.1.2 The Stem classes

class inpystem.signals.**AbstractStem**(*hsdata*, *scan=None*, *verbose=True*)

Abstract STEM acquisition class.

This is an *abstract* class, which mean you can not instantiate such object.

It defines the structure for a STEM acquisition object.

Variables

- **hsdata** (*hs BaseSignal*) – The acquired STEM data hyperspy object.
- **scan** (*Scan object*) – The sampling scan object associated with the data.
- **verbose** (*bool*) – If True, information is sent to standard output. Default is True.

__init__(*hsdata*, *scan=None*, *verbose=True*)

AbstractStem constructor.

Parameters

- **hsdata** (*hs BaseSignal*) – The acquired STEM data hyperspy object.
- **scan** (*optional, Scan object*) – The sampling scan object associated with the data. Default is None for full sampling.
- **verbose** (*bool*) – If True, information is sent to standard output. Default is True.

correct(*rows=slice(None, None, None)*, *cols=slice(None, None, None)*, *bands=slice(None, None, None)*, *dpixels=None*)

Correct deffective data.

Deffective data correspond to:

1. Rows to remove at the begging or at the end of the image.
2. Columns to remove at the begging or at the end of the image.
3. Bands to remove at the begging or at the end of the image.
4. Located dead pixels at the center of the image.

In the cases 1, 2 or 3, the rows and columns are purely removed. The dead pixels are filled with the mean over a neighbourhood.

A :code‘slice‘ object for an object `A` of length `L` defines a continuous portion of `A` such as `A[n_1], A[n_1+1], ..., A[n_2-1]` with `n_1 < n_2`. In such case, a slice object definition is `slice(n_1, n_2)`. If `n_1` is 0, then use `slice(None, n_2)`. If `n_2` is `L` use `slice(n_1, None)`. Last, if all the elements of `A` should be kept, use `slice(None)`.

Parameters

- **rows** (*slice object*) – The range of rows to keep.
- **cols** (*slice object*) – The range of columns to keep.
- **cols** – The range of bands to keep.

- **dpixels** (*list*) – The positions of the dead pixels.

correct_fromfile (*file*, *force_ndim=None*)
force_ndim aims at forcing dimension. Default is None for data dimension.

abstract restore ()
Restores corrupted data.

plot ()
Plots the masked data.

class inpystem.signals.**Stem2D** (*hsdata*, *scan=None*, *verbose=True*)
2D image STEM acquisition.

This defines a 2D STEM image with its associated sampling scan.

Variables

- **hsdata** (*hs BaseSignal*) – The acquired STEM data hyperspy object.
- **scan** (*Path object*) – The sampling scan object associated with the data.

restore (*method='interpolation'*, *parameters={}*)
Restores the acquisition.

It performs denoising in the case of full scan and performs recontruction in case of partial sampling.

class inpystem.signals.**Stem3D** (*hsdata*, *scan=None*, *verbose=True*)
3D image STEM acquisition.

This defines a 3D STEM image with its associated sampling scan.

Variables

- **hsdata** (*hs BaseSignal*) – The acquired STEM data hyperspy object.
- **scan** (*Path object*) – The sampling scan object associated with the data.

restore (*method='interpolation'*, *parameters={}*, *PCA_transform=True*, *PCA_th='auto'*)
Restores the acquisition.

It performs denoising in the case of full scan and performs recontruction in case of partial sampling.

Parameters

- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.

plot_sum ()
Shows the sum of the data along the last axis.

plot_as2D ()
Implements the HypersSpy tool to visualize the image for a given band.

plot_as1D ()
Implements the HypersSpy tool to visualize the spectrum for a given pixel.

plot_roi ()
Implements the Hyperspy tool to analyse regions of interest.

3.2 dev module

This module defines the basic stem acquisitions objects used for processing. These objects are:

1. The 3D spectrum-image,
2. The 2D HAADF image,

class `inpystem.dev.`**`AbstractDev`**(*key*, *data*, *mask=None*, *sigma=None*, *seed=None*, *normalize=True*, *verbose=True*)

Abstract Dev acquisition class.

This is an *abstract* class, which mean you can not instantiate such object.

It defines the structure for a Dev acquisition object.

key: str 1-word description of the Dev2D image.

data: (m,n) or (m, n, l) numpy array The Dev2D image data before the noise step. Its dimension is (m,n).

ndata: (m,n) or (m, n, l) numpy array The noised Dev2D image. If `snr` is None, `ndata` is None. Its dimension is (m,n).

sigma: float The noise standard deviation.

seed: optional, int The random noise matrix seed.

normalize: bool If :code:normalize' is True, the data will be centered and normalize before the corruption steps.

mean_std: None, 2-tuple It stores the data mean and std in case normalize is True.

verbose: bool If True, information will be displayed. Default is True.

`__init__`(*key*, *data*, *mask=None*, *sigma=None*, *seed=None*, *normalize=True*, *verbose=True*)

AbstractDev constructor.

Parameters

- **key** (*str*) – 1-word description of the Dev2D image. Generally, it's common to the stem acquisition object.
- **data** (*(m, n) or (m, n, l) numpy array*) – The noise-free image data.
- **mask** (*(m, n) numpy array*) – The sampling mask.
- **sigma** (*optional, None, float*) – The desired standard deviation used to model noise. Dafault is None for no additional noise.
- **seed** (*optional, None, int*) – The random noise matrix seed. Dafault is None for no seed initialization.
- **normalize** (*optional, bool*) – If :code:normalize' is True, the data will be centered and normalize before the corruption steps. Default is True.
- **verbose** (*optional, bool*) – If True, information will be displayed. Default is True.

property seed

seed property getter.

property sigma

sigma property getter.

`set_ndata`()

Constructs the noised data.

It is also used to draw a new noise matrix.

class inpystem.dev.**Dev2D**(*key*, *hsdata*, *scan=None*, *modif_file=None*, *sigma=None*, *seed=None*, *normalize=True*, *verbose=True*)

Dev2D Class.

Variables

- **key** (*str*) – 1-word description of the Dev2D image.
- **hsdata** (*Signal2D hyperspy data*) – The hyperspy Signal2D image. Its dimension is denoted (m,n). This is used to communicate with the parrent class.
- **data** (*(m,n) numpy array*) – The Dev2D image data before the noise step. Its dimension is (m,n).
- **ndata** (*(m,n) numpy array*) – The noised Dev2D image. If `snr` is None, `ndata` is None. Its dimension is (m,n).
- **scan** (*optional, Scan object*) – The sampling scan object associated with the data. Default is None for full sampling.
- **sigma** (*float*) – The noise standard deviation.
- **seed** (*optional, int*) – The random noise matrix seed.
- **normalize** (*bool*) – If :code:normalize‘ is True, the data will be centered and normalize before the corruption steps.
- **mean_std** (*None, 2-tuple*) – It stores the data mean and std in case normalize is True.
- **verbose** (*bool*) – If True, information will be displayed. Default is True.

__init__(*key*, *hsdata*, *scan=None*, *modif_file=None*, *sigma=None*, *seed=None*, *normalize=True*, *verbose=True*)

SpectrumImage constructor.

Parameters

- **key** (*str*) – 1-word description of the Dev2D image. Generally, it's common to the stem acquisition object.
- **hsdata** (*Signal2D hyperspy data*) – The noise-free Dev2D image data. Its dimension is denoted (m,n).
- **scan** (*optional, None, Scan object*) – The sampling scan object associated with the data. Default is None for full sampling.
- **modif_file** (*optional, None, str*) – A .conf configuration file to remove rows, columns or dead pixels. Default is None for no modification.
- **sigma** (*optional, None, float*) – The desired standard deviation used to model noise. Dafault is None for no additional noise.
- **seed** (*optional, None, int*) – The random noise matrix seed. Dafault is None for no seed initialization.
- **normalize** (*optional, bool*) – If :code:normalize‘ is True, the data will be centered and normalize before the corruption steps. Default is True.
- **verbose** (*optional, bool*) – If True, information will be displayed. Default is True.

restore(*method='interpolation'*, *parameters={}*, *verbose=None*)

plot(*noised=False*)

Plots the haadf image.

Parameters **noised** (*optional, bool*) – If True, the noised data is used. If False, the noise-free data is shown. Default is False.

class inpystem.dev.**Dev3D**(*key*, *hsdata*, *scan=None*, *modif_file=None*, *sigma=None*, *seed=None*, *normalize=True*, *PCA_transform=False*, *PCA_th='auto'*, *verbose=True*)

Dev3D Class

Variables

- **key** (*str*) – 1-word description of the Dev3D.
- **hsdata** (*Signal1D hyperspy data*) – The hyperspy Signal2D image. Its dimension is denoted (m,n). This is used to communicate with the parrent class.
- **data** (*(m,n,l) numpy array*) – The Dev3D data before the noise step. Its dimension is (m,n,l).
- **ndata** (*(m,n,l) numpy array*) – The noised Dev3D data. If snr is None, ndata is None. Its dimension is (m,n,l).
- **snr** (*optional, float*) – The desired snr used for the noising step.
- **sigma** (*float*) – The noise standard deviation.
- **seed** (*optional, int*) – The random noise matrix seed.
- **normalize** (*bool*) – If normalize is True, the data will be centered and normalize before the corruption steps.
- **PCA_transform** (*bool*) – If PCA_transformed is True, a PCA transformation has been applied to the data.
- **PCA_info** (*None, dictionary*) – If PCA_transformed is True, PCA_info contains informations about the reduction. Otherwise, it is None.
- **PCA_operator** (*PcaHandler*) – The PCA operator.
- **verbose** (*bool*) – If True, information will be displayed. Default is True.

__init__(*key*, *hsdata*, *scan=None*, *modif_file=None*, *sigma=None*, *seed=None*, *normalize=True*, *PCA_transform=False*, *PCA_th='auto'*, *verbose=True*)

Dev3D __init__ function.

Parameters

- **key** (*str*) – 1-word description of the Dev3D image. Generally, it's common to the stem acquisition object.
- **hsdata** (*Signal1D hyperspy data*) – The noise-free Dev3D image data. Its dimension is denoted (m,n, l).
- **scan** (*optional, None, Scan object*) – The sampling scan object associated with the data. Default is None for full sampling.
- **modif_file** (*optional, None, str*) – A .conf configuration file to remove rows, columns or dead pixels. Default is None for no modification.
- **sigma** (*optional, None, float*) – The desired standard deviation used to model noise. Dafault is None for no additional noise.
- **seed** (*optional, None, int*) – The random noise matrix seed. Dafault is None for no seed initialization.
- **normalize** (*optional, bool*) – If :code:normalize‘ is True, the data will be centered and normalize before the corruption steps. Default is True.

- **PCA_transform** (*optional, bool*) – If PCA_transformed is True, a PCA transformation is applied to the data. Default is False.
- **PCA_th** (*optional, str, int*) – The desired data dimension after dimension reduction. Possible values are:
  - 'auto' for automatic choice,
  - 'max' for maximum value
  - an int value for user value.

  Default is 'auto'.
- **verbose** (*optional, bool*) – If True, information will be displayed. Default is True.

direct_transform(*data*)

Applies the Dev3D PCA transformation and normalization steps to data.

Parameters **data** (*(m, n, l) numpy array, hs image*) – Data whose shape is the same as self.data.

inverse_transform(*data*)

Applies the Dev3D PCA inverse transformation and inverse normalization steps to spim.

Parameters **data** (*(m, n, l) numpy array, hs image*) – Data whose shape is the same as self.data.

restore(*method='interpolation', parameters={}, PCA_transform=None, PCA_th='auto', verbose=None*)

show_sum(*noised=False*)

Shows the sum of the data along the last axis.

Parameters **noised** (*optional, bool*) – If True, the noised data is used. If False, the noise-free data is shown. Default is False.

plot_as2D(*noised=False*)

Implements the HypersSpy tool to visualize the image for a given band.

Parameters **noised** (*optional, bool*) – If True, the noised data is used. If False, the noise-free data is shown. Default is False.

plot_as1D(*noised=False*)

Implements the HypersSpy tool to visualize the spectrum for a given pixel.

Parameters **noised** (*optional, bool*) – If True, the noised data is used. If False, the noise-free data is shown. Default is False.

plot_roi(*noised=False*)

Implements the Hyperspy tool to analyse regions of interest.

Parameters **noised** (*optional, bool*) – If True, the noised data is used. If False, the noise-free data is shown. Default is False.

3.3 dataset module

This module defines important tools to import data from dataset.

3.3.1 Data path functions

`inpystem.dataset.`**`read_data_path`**`()`

Read the saved data folder path.

The inpystem library proposes to store all data in a particular directory with associated configuration files. This folder is saved in inpystem. To access to this folder path, use this function.

If no data path is saved, the function returns None. Else, the path is returned.

Returns None is returned if no path is saved. Else, the data path is returned.

Return type None, str

`inpystem.dataset.`**`set_data_path`**`(`*path*`)`

Sets the saved data folder path.

The inpystem library proposes to store all data in a particular directory with associated configuration files. This folder is saved in inpystem. To set to this folder path, use this function.

A boolean is returned to confirm that the change is effective.

Parameters **`path`** (*`str`*) – The desired data path.

Returns If the data path has really been changed, the function returns True. Else, it returns False.

Return type bool

3.3.2 Load functions

`inpystem.dataset.`**`load_file`**`(`*file*, *ndim*, *scan_ratio=None*, *scan_seed=None*, *dev=None*, *verbose=True*`)`

This function loads a STEM acquisition based on a configuration .conf file path.

The number of dimensions ndim should also be given.

The Path is generated from a scan file given in the configuration file or is randomly drawn. Whatever the case, the Scan object `ratio` property can be set through the `scan_ratio` argument. Additionally, in the case where no file is provided for the scan pattern, use the `scan_seed` argument to have reproductible data.

The function allows the user to ask for development data by setting the `dev` argument. If `dev` is None, then the usual Stem2D and Stem3D classes are returned. If `dev` is a dictionary, then Dev2D and Dev3D classes are returned. This dictionary could contain additional class arguments such as:

- snr, seed and normalized for Dev2D,
- snr, seed, normalized, PCA_transformed and PCA_th for Dev3D.

Parameters

- **`file`** (*`str`*) – The configuration file path.
- **`ndim`** (*`int`*) – The data dimension. Should be 2 or 3.
- **`scan_ratio`** (*`optional, None, float`*) – The Path object ratio. Default is None for full sampling.
- **`scan_seed`** (*`int`*) – The seed in case of random scan initialization. Default is None for random seed.
- **`dev`** (*`optional, None, dictionary`*) – This arguments allows the user to ask for development data. If this is None, usual data is returned. If this argument is a dictionary, then

development data will be returned and the dictionary will be given to the data contructors. Default is None for usual data.

- **verbose** (*optional, bool*) – If True, information will be sent to standard output.. Default is True.

Returns The inpystem data.

Return type *Stem2D*, *Stem3D*, *Dev2D*, *Dev3D*

Todo: Maybe enable PCA_th in config file for 3D data.

`inpystem.dataset.`**`load_key`**(*key*, *ndim*, *scan_ratio=None*, *scan_seed=None*, *dev=None*, *verbose=True*)

This function loads a STEM acquisition based on a key.

A key is a string which can be:

- an example data name,
- the name of some data located in the inpystem data path (which is defined with the `inpystem.dataset.set_data_path()` function).

The key should always be the name of the configuration file without the suffix (.conf). As an example, if a configuration file located in the data folder is named my-sample.conf, then its data could be loaded with the my-sample key.

The number of dimensions ndim should also be given.

The Path is generated from a scan file given in the configuration file or is randomly drawn. Whatever the case, the Scan object `ratio` property can be set through the `scan_ratio` argument. Additionally, in the case where no file is provided for the scan pattern, use the `scan_seed` argument to have reproductible data.

The function allows the user to ask for development data by setting the `dev` argument. If `dev` is None, then the usual Stem2D and Stem3D classes are returned. If `dev` is a dictionary, then Dev2D and Dev3D classes are returned. This dictionary could contain additional class arguments such as:

- snr, seed, normalized and verbose for Dev2D,
- snr, seed, normalized, PCA_transformed, PCA_th and verbose for Dev3D.

This function only searches for the configuration file to use the load_file function afterwards.

Parameters

- **key** (*str*) – The data key.
- **ndim** (*int*) – The data dimension. Should be 2 or 3.
- **scan_ratio** (*optional, None, float*) – The Path object ratio. Default is None for full sampling.
- **scan_seed** (*int*) – The seed in case of random scan initialization. Default is None for random seed.
- **dev** (*optional, None, dictionary*) – This arguments allows the user to ask for development data. If this is None, usual data is returned. If this argument is a dictionary, then development data will be returned and the dictionary will be given to the data contructors. Default is None for usual data.
- **verbose** (*optional, bool*) – If True, information will be sent to standard output.. Default is True.

Returns The inpystem data.

Return type *Stem2D*, *Stem3D*, *Dev2D*, *Dev3D*

3.4 inpystem.tools subpackage

This introduces some tools for inpystem.

3.4.1 inpystem.tools.PCA module

This module implements tools to perform PCA transformation.

The main element is the **PcaHandler** class which is a user interface. It performs direct and inverse PCA transformation for 3D data.

Dimension_Reduction is the background function which performs PCA while the **EigenEstimate** function improves the estimation of PCA eigenvalues.

The PcaHandler interface

class `inpystem.tools.PCA.`**`PcaHandler`**(*Y*, *mask=None*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)

Interface to perform PCA.

The PCA is applied at class initialization based on the input data. This same operation can be applied afterward to other data using the `direct` and `inverse` methods.

Variables

- **`Y`** (*`(m, n, l) numpy array`*) – Multi-band data.
- **`Y_PCA`** (*`(m, n, PCA_th) numpy array`*) – The data in PCA space.
- **`mask`** (*`optional, (m, n) numpy array`*) – Spatial sampling mask. Default is full sampling.
- **`PCA_transform`** (*`optional, bool`*) – Flag that sets if PCA should really be applied. This is useful in soma cases where PCA has already been applied. Default is True.
- **`verbose`** (*`optional, bool`*) – If True, information is sent to output.
- **`H`** (*`(l, PCA_th) numpy array`*) – The subspace base.
- **`Ym`** (*`(m, n, l) numpy array`*) – Matrix whose spectra are all composed of the data spectral mean.
- **`PCA_th`** (*`int`*) – The estimated data dimension.
- **`InfoOut`** (*`dict`*) – The dictionary contaning additional information about the reduction. See Note.

Note: The InfoOut dictionary containg the thee following keys:

1. 'H' which is the base of the reduced subspace. Its shape is (l, PCA_th) where PCA_th is the estimated data dimension.
2. 'd' which is the evolution of the PCA-eigenvalues after estimation.
3. 'PCA_th' which is the estimated data dimension.

4. 'sigma' which is the estimated Gaussian noise standard deviation.
5. 'Ym' which is a (m, n, l) numpy array where the data mean over bands is repeated for each spatial location.

`__init__`(*Y*, *mask=None*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)
PcaHandler constructor.

Parameters

- **`Y`** (`(m, n, l) numpy array`) – Multi-band data.
- **`mask`** (`(m, n) numpy array`) – Spatial sampling mask.
- **`PCA_transform`** (`optional, bool`) – Flag that sets if PCA should really be applied. This is useful in soma cases where PCA has already been applied. Default is True.
- **`verbose`** (`optional, bool`) – If True, information is sent to output.

`direct`(*X=None*)
Performs direct PCA transformation.

The input X array can be data to project into the PCA subspace or None. If input is None (which is default), the output will be simply self.Y_PCA.

> **Caution:** The input data to transform should have the same shape as the Y initial data.

Parameters **`X`** (`(m, n, l) numpy array`) – The data to transform into PCA space.

Returns Multi-band data in reduced space.

Return type (m, n, PCA_th) numpy array

`inverse`(*X_PCA*)
Performs inverse PCA transformation.

> **Caution:** The input data to transform should have the same shape as the self.Y_PCA transformed data.

Parameters **`X_PCA`** (`(m, n, PCA_th) numpy array`) – The data to transform into data space.

Returns Multi-band data after inverse transformation.

Return type (m, n, l) numpy array

Backgroud functions

`inpystem.tools.PCA.`**`Dimension_Reduction`**(*Y*, *mask=None*, *PCA_th='auto'*, *verbose=True*)
Reduces the dimension of a multi-band image.

Parameters

- **`Y`** (`(m, n, l) numpy array`) – The multi-band image where the last axis is the spectral one.

- **mask** (*optional, (m, n) numpy array*) – The spatial sampling mask filled with True where pixels are sampled. This is used to remove correctly the data mean. Default if a matrix full of True.
- **PCA_th** (*optional, str, int*) – The PCA threshold. 'auto' for automatic estimation. 'max' to keep all components. An interger to choose the threshold. In case there are less samples (N) than the data dimension (l), thi sparameter is overridded to keep a threshold of N-1.
- **verbose** (*optional, bool*) – Prints output if True. Default is True.

Returns

- *(m, n, PCA_th) numpy array* – The data in the reduced subspace. Its shape is (m, n, PCA_th) where PCA_th is the estimated data dimension.
- *dict* – The dictionary contaning additional information about the reduction. See Note.

Note: The InfoOut dictionary containg the thee following keys:

1. 'H' which is the base of the reduced subspace. Its shape is (l, PCA_th) where PCA_th is the estimated data dimension.
2. 'd' which is the evolution of the PCA-eigenvalues after estimation.
3. 'PCA_th' which is the estimated data dimension.
4. 'sigma' which is the estimated Gaussian noise standard deviation.
5. 'Ym' which is a (m, n, l) numpy array where the data mean over bands is repeated for each spatial location.

`inpystem.tools.PCA.`**`EigenEstimate`**(*l*, *Ns*)

Computes an estimate of the covariance eigenvalues given the sample covariance eigenvalues. The Stein estimator coupled with isotonic regression has been used here.

For more information, have a look at:

-
- MESTRE, Xavier. Improved estimation of eigenvalues and eigenvectors of covariance matrices using their sample estimates. IEEE Transactions on Information Theory, 2008, vol. 54, no 11, p. 5113-5129.s

Parameters

- **l** (*numpy array*) – Sample eigenvalues
- **Ns** (*int*) – Number of observations

Returns

- *numpy array* – Estimated covariance matrix eigenvalues.
- *float* – Estimated Gaussian noise standard deviation.
- *int* – Estimated dimension of the signal subspace.

3.4.2 inpystem.tools.FISTA module

This module implements interfacing tools for the FISTA algorithm.

For further informations about the FISTA algorithm, have a look at[1].

class `inpystem.tools.FISTA.`**`FISTA`**(*f*, *df*, *L*, *g*, *pg*, *shape*, *Nit=None*, *init=None*, *verbose=True*)

Fast Iterative Shrinkage-Thresholding Algorithm implementation.

Variables

- **`f`** (*`function`*) – $C^{1,1}$ convex function.
- **`df`** (*`function`*) – derivative function of f.
- **`L`** (*`float`*) – Lipshitz contant of f.
- **`g`** (*`function`*) – Non-smooth function.
- **`pg`** (*`function`*) – g poximal operator.
- **`shape`** (*`tuple`*) – The data shape.
- **`Nit`** (*`None, int`*) – Number of iteration. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **`init`** (*`numpy array`*) – Init point which shape is the same as the data. If None, a random initailization is drawn. Default is None.
- **`verbose`** (*`bool`*) – If True, process informations are sent to the output. Default is True.
- **`Nit_max`** (*`int`*) – Maximum number of iterations.
- **`tau`** (*`float`*) – Descent step.
- **`E`** (*`numpy array`*) – Functional evolution across iterations.
- **`lim`** (*`float`*) – Controlls the stop condition in case Nit is None. The smallest lim, the more iterations before stopping. lim is usually 1e-4.

`__init__`(*f*, *df*, *L*, *g*, *pg*, *shape*, *Nit=None*, *init=None*, *verbose=True*)

Initialization function for FISTA.

Parameters

- **`f`** (*`function`*) – $C^{1,1}$ convex function.
- **`df`** (*`function`*) – derivative function of f.
- **`L`** (*`float`*) – Lipshitz contant of f.
- **`g`** (*`function`*) – Non-smooth function.
- **`pg`** (*`function`*) – g poximal operator.
- **`shape`** (*`tuple`*) – The data shape.
- **`Nit`** (*`None, int`*) – Number of iteration. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **`init`** (*`numpy array`*) – Init point which shape is the same as the data. If None, a random initailization is drawn. Default is None.
- **`verbose`** (*`bool`*) – If True, process informations are sent to the output. Default is True.

`StopCritera`(*n*)

This function computes a critera that informs about the algorithm convergence at step n.

Parameters **`n`** (*`int`*) – Current step

[1] BECK, Amir et TEBOULLE, Marc. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2009, vol. 2, no 1, p. 183-202.

Returns Value of the critera.

Return type float

StopTest (*n*)

This function choose if iterations should be stopped at step n. If Nit is not None, it returns True as long as n is smaller than Nit. If Nit is None, it returns True as long as the functional is evolving fast.

Parameters **n** (*int*) – Current step.

Returns Should the iterations go on ?

Return type bool

execute ()

Method that executes the FISTA algorithm.

Returns

- *numpy array* – The optimum of the optimization problem.
- *dict* – Extra informations about convergence.

Note: Infos in output dictionary:

- `E`: Evolution of the functional along the iterations.
- `time`: Execution time.

3.4.3 inpystem.tools.matlab_interface module

This module defines an interface to run matlab codes from python.

inpystem.tools.matlab_interface.**matlab_interface** (*program*, *dataDico*)

Interfaces a matlab code with python3.

The functions needs a **matlab program** to run and **input data** to be given to the matlab program.

The input data should be given in dictionary format where keys are the matlab variable names and values are the variable data.

Parameters

- **program** (*str, Path object*) – The program path.
- **dataDico** (*dict*) – The dico containing the data to give to the program.

Returns The data returned by the program.

Return type dict

Note: A matlab command *matlab* should be accessible in the command line to make this code work.

If this does not work, please be sure the PATH variable is perfecty set. For exemple, please add this to your *.bashrc* for Linux Users:

Listing 1: .bashrc

```
export PATH:$PATH:/path/to/matlab/bin
```

and for Windows users, please have a search about how to add a location into your path (this is a graphical task).

3.4.4 inpystem.tools.dct module

This module defines some functions related to DCT decomposition including:

- direct and inverse normalized 2D DCT transform,
- direct and inverse band-by-band DCT transform for multi-band data.

2D transformations

`inpystem.tools.dct.`**`dct2d`**`(`*a*`)`

Computes the 2D Normalized DCT-II.

Parameters **`X`** (*`(m, n) numpy array`*) – 2D image.

Returns DCT coefficient matrix.

Return type (m, n) numpy array

`inpystem.tools.dct.`**`idct2d`**`(`*a*`)`

Computes the 2D Normalized Inverse DCT-II.

Parameters **`A`** (*`(m, n) numpy array`*) – DCT coefficient matrix

Returns 2D image.

Return type (m, n) numpy array

Band-by-band transformations

`inpystem.tools.dct.`**`dct2d_bb`**`(`*x*, *shape=None*`)`

Computes the band-by-band 2D Normalized DCT-II

If the input X is a 3D data cube, the 2D dct will be computed for each 2D images staked along the 2nd axis.

Parameters

- **`X`** (*`(l, m*n) or (m, n, l) numpy array`*) – 2D or 3D multi-band data. If the data has 3 dimensions, the last axis is for spetra. If the data is 2D, the first axis is for spectra.
- **`shape`** (*`optional, (m, n, l) tuple`*) – This is the data shape. This parameter is required only if input data are 2D.

Returns DCT coefficient matrix.

Return type (l, m*n) or (m, n, l) numpy array

`inpystem.tools.dct.`**`idct2d_bb`**`(`*a*, *shape=None*`)`

Computes the band-by-band inverse 2D Normalized DCT-II

If the input a is a 3D data cube, the 2D dct will be computed for each 2D images staked along the 2nd axis.

Parameters

- **`A`** (*`(l, m*n) or (m, n, l) numpy array`*) – 2D or 3D multi-band data DCT decomposition. If the data has 3 dimensions, the last axis is for spetra. If the data is 2D, the first axis is for spectra.
- **`shape`** (*`optional, (m, n, l) tuple`*) – This is the data shape. This parameter is required only if input data are 2D.

Returns The image matrix.

Return type (l, m*n) or (m, n, l) numpy array

3.4.5 inpystem.tools.sec2str module

This small module only contain sec2str, which is a function to display time in human-readable format.

`inpystem.tools.sec2str.`**`sec2str`**(*t*)

Returns a human-readable time str from a duration in s.

Parameters **t** (*float*) – Duration in seconds.

Returns Human-readable time str.

Return type str

Example

```
>>> from inpystem.tools.sec2str import sec2str
>>> sec2str(5.2056)
5.21s
>>> sec2str(3905)
'1h 5m 5s'
```

3.4.6 inpystem.tools.metrics module

This module contains several metric functions.

`inpystem.tools.metrics.`**`SNR`**(*xhat*, *xref*)

Computes the SNR metric.

Parameters

- **xhat** (*numpy array*) – The noised data.
- **xref** (*numpy array*) – The noise-free image.

Returns The SNR value in dB.

Return type float

`inpystem.tools.metrics.`**`NMSE`**(*xhat*, *xref*)

Computes the normalized mean square metric.

Parameters

- **xhat** (*numpy array*) – The noised data.
- **xref** (*numpy array*) – The noise-free image.

Returns The NMSE value.

Return type float

`inpystem.tools.metrics.`**`aSAD`**(*xhat*, *xref*)

Computes the averaged Spectral Angle Distance metric.

The input data number of dimensions can be:

- 1: the data are spectra,
- 2: the data are matrices of shape (n, M),

- 3: the data are matrices of shape (m, n, M)

where M is the spectrum size.

Parameters

- **`xhat`** (*`numpy array`*) – The noised data.
- **`xref`** (*`numpy array`*) – The noise-free image.

Returns The (mean) aSAD value.

Return type float

`inpystem.tools.metrics.`**`SSIM`**(*xhat*, *xref*)

Computes the structural similarity index.

Parameters

- **`xhat`** (*`numpy array`*) – The noised data.
- **`xref`** (*`numpy array`*) – The noise-free image.

Returns The (mean) SSIM value.

Return type float

3.4.7 inpystem.tools.misc module

This module defines some miscellaneous functions.

`inpystem.tools.misc.`**`toslice`**(*text=None*, *length=None*)

Parses a string into a slice.

Input strings can be eg. '5:10', ':10', '1:'. Negative limits are allowed only if the data length is given. In such case, input strings can be e.g. '1:-10'. Last, an integer can be given alone such as '1' to select only the 1st element.

If no text not length is given, default slice is slice(None).

Parameters

- **`text`** (*`optional, None, str`*) – The input text to parse. Default is None.
- **`length`** (*`None, int`*) – The data length. This is not mendatory if no slice limit is negative. Dafault is None.

Returns The parsed slice object.

Return type slice

3.5 inpystem.restore subpackage

This package implements a variety of reconstruction algorithms for 2D as 3D data.

These methods include:

- interpolation methods that are known to be fast but with low-quality results,
- regularized least-square methods which are a slower than interpolation but with higher quality,
- dictionary-learning methods which are very efficient at the price of long reconstruction procedures.

3.5.1 inpystem.restore.interpolation module

This module implement the interpolate function which is an interface between 2D as 3D data and the interpolation function of scipy.

`inpystem.restore.interpolation.`**`interpolate`**`(`*Y*, *mask=None*, *method='nearest'*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*`)`

Implements data interpolation.

Three interpolation methods are implemented: nearest neighbor, linear interpolation and cubic interpolation.

Note that cubic interpolation is performed band-by-band in the case of 3D data while other methods perform in 3D directly.

Parameters

- **`Y`** (*`(m, n) or (m, n, l) numpy array`*) – Input data
- **`mask`** (*`optional, None, (m, n) numpy array`*) – Sampling mask. True for sampled pixel. Default is None for full sampling.
- **`method`** (*`optional, 'nearest', 'linear' or 'cubic'`*) – Interpolation method. Default is 'nearest'.
- **`PCA_transform`** (*`optional, bool`*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **`PCA_th`** (*`optional, int, str`*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **`verbose`** (*`optional, bool`*) – Indicates if information text is desired. Default is True.

Returns

- *(m, n) or (m, n, l) numpy array* – Interpolated data.
- *float* – Execution time (s).

3.5.2 inpystem.restore.LS_2D module

This module gathers regularized least square restoration methods adapted to 2D data.

The only method it implements for the moment is the **L1-LS** algorithm.

`inpystem.restore.LS_2D.`**`L1_LS`**`(`*Y*, *Lambda*, *mask=None*, *init=None*, *Nit=None*, *verbose=True*`)`

L1-LS algorithm.

The L1-LS algorithm denoises or reconstructs an image possibly spatially sub-sampled in the case of spatially sparse content in the DCT basis. It is well adapted to periodic data.

This algorithms solves the folowing regularization problem:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}\in\mathbb{R}^{m\times n}} \frac{1}{2}||(\mathbf{x}-\mathbf{y})\cdot\Phi||_F^2 + \lambda||\mathbf{x}\Psi||_1$$

where $\mathbf{y}$ are the corrupted data, Φ is a subsampling operator and Ψ is a 2D DCT operator.

Caution: It is strongly recomended to remove the mean before reconstruction. Otherwise, this value could be lost automaticaly by the algorithm in case of powerful high frequencies.

In the same way, normalizing the data is a good practice to have the parameter be low sensitive to data.

These two operations are implemented in this function.

Parameters

- **`(m, n) numpy array`** (*`Y`*) – An image which mean has been removed.
- **`Lambda`** (*`float`*) – Regularization parameter.
- **`mask`** (*`optional, None, (m, n) numpy array`*) – A sampling mask which is True if the pixel is sampled. Default is None for full sampling.
- **`init`** (*`optional, None, (m, n, l) numpy array`*) – The algorithm initialization. Default is None for random initialization.
- **`Nit`** (*`optional, None, int`*) – Number of iteration in case of inpainting. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **`verbose`** (*`optional, bool`*) – Indicates if information text is desired. Default is True.

Returns

- *(m, n) numpy array* – The reconstructed/denoised image.
- *dict* – A dictionary containing some extra info

Note: Infos in output dictionary:

- `E`: In the case of partial reconstruction, the cost function evolution over iterations.
- `Gamma`: The array of kept coefficients (order is Fortran-style).
- `nnz_ratio`: the ratio Gamma.size/(m*n).

3.5.3 inpystem.restore.LS_3D module

This module implements regularized least square restoration methods adapted to 3D data.

The two methods it gathers are

- **Smoothed SubSpace (3S) algorith**,
- **Smoothed Nuclear Norm (SNN) algorithm**.

`inpystem.restore.LS_3D.`**`SSS`**(*Y*, *Lambda*, *mask=None*, *PCA_transform=True*, *PCA_th='auto'*, *PCA_info=None*, *scale=1*, *init=None*, *Nit=None*, *verbose=True*)

Smoothed SubSpace algorithm.

The 3S algorithm denoise or reconstructs a multi-band image possibly spatially sub-sampled in the case of spatially smooth images. It is well adapted to intermediate scale images.

This algorithm performs a PCA pre-processing operation to estimate:

- the data subspace basis **H**,
- the subspace dimension R,

- the associated eigenvalues in decreasing order $\mathbf{d}$,
- the noise level $\hat{\sigma}$.

After this estimation step, the algorithm solves the folowing regularization problem in the PCA space:

$$\hat{\mathbf{S}} = \underset{\mathbf{S}\in\mathbb{R}^{m\times n\times R}}{\arg\min} \frac{1}{2R}\|\mathbf{S}\mathbf{D}\|_\mathrm{F}^2 + \frac{\lambda}{2}\sum_{m=1}^{R} w_m |\mathbf{S}_{m,:}|_2^2$$
$$\text{s.t.} \quad \frac{1}{R}|\mathbf{H}_{1:R}^T\mathbf{Y}_{\mathcal{I}(n)} - \mathbf{S}_{\mathcal{I}(n)}|_2^2 \leq \hat{\sigma}^2,\ \forall n \in \{1,\ldots,\ m*n\}$$

where $\mathbf{Y}$ are the corrupted data, $\mathbf{D}$ is a spatial finite difference operator and $\mathcal{I}$ is the set of all sampled pixels.

Caution: It is strongly recomended to perform PCA before running the algorithm core. This operation is integrated in this function.

In case this pre-processing step has already been done, set the `PCA_transform` parameter to False to disable the PCA step included in the SSS function. If `PCA_transform` is set to False, the `PCA_info` parameter is required.

Parameters

- **`(m, n, l) numpy array`** (*`Y`*) – A 3D multi-band image.
- **`Lambda`** (*`float`*) – Regularization parameter.
- **`mask`** (*`optional, None, (m, n) numpy array`*) – A sampling mask which is True if the pixel is sampled. Default is None for full sampling.
- **`PCA_transform`** (*`optional, bool`*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **`PCA_th`** (*`optional, int, str`*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **`PCA_info`** (*`optional, dict`*) – In case PCA_transform is False, some extra info should be given to SSS. The required keys for PCA_info are:
  - 'd' which are the PCA eigenvalues.
  - 'sigma' which is an estimate of the data noise std.
- **`scale`** (*`optional, float`*) – Scales the prox operator sphere radius. Should lay in]0, +inf[. Default is 1.
- **`init`** (*`optional, None, (m, n, l) numpy array`*) – The algorithm initialization. Default is None for random initialization.
- **`Nit`** (*`optional, None, int`*) – Number of iteration in case of inpainting. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **`verbose`** (*`optional, bool`*) – Indicates if information text is desired. Default is True.

Returns

- *(m, n, l) numpy array* – The reconstructed/denoised multi-band image.
- *dict* – A dictionary containing some extra info

Note: Infos in output dictionary:

- `E`: in the case of partial reconstruction, the cost function evolution over iterations.
- `H` : the basis of the chosen signal subspace

References

Monier, E., Oberlin, T., Brun, N., Tencé, M., de Frutos, M., & Dobigeon, N. (2018). Reconstruction of Partially Sampled Multiband Images—Application to STEM-EELS Imaging. IEEE Trans. Comput. Imag., 4(4), 585–598.

`inpystem.restore.LS_3D.`**`SNN`**`(`*Y*, *Lambda*, *Mu*, *mask=None*, *PCA_transform=True*, *PCA_th='auto'*, *init=None*, *Nit=None*, *verbose=True*`)`

Smoothed Nuclear Norm algorithm.

The SNN algorithm denoise or reconstructs a multi-band image possibly spatially sub-sampled in the case of spatially smooth images. It is well adapted to intermediate scale images.

This algorithm solves the folowing optimization problem:

$$\hat{\mathbf{X}} = \underset{\mathbf{X}\in\mathbb{R}^{m\times n\times B}}{\arg\min} \frac{1}{2}||\mathbf{Y}_\mathcal{I} - \mathbf{X}_\mathcal{I}||_\mathrm{F}^2 + \frac{\lambda}{2}\|\mathbf{X}\mathbf{D}\|_\mathrm{F}^2 + \mu||\mathbf{X}||_*$$

where $\mathbf{Y}$ are the corrupted data, $\mathbf{D}$ is a spatial finite difference operator and $\mathcal{I}$ is the set of all sampled pixels.

This algorithm can perform a PCA pre-processing operation to estimate:

- the data subspace basis $\mathbf{H}$,
- the subspace dimension R.

This is particularly usefull to reduce the data dimension and the execution time and to impose a data low-rank property.

Caution: It is strongly recomended to perform PCA before running the algorithm core. This operation is integrated in this function.

In case this pre-processing step has already been done, set the `PCA_transform` parameter to False to disable the PCA step included in the CLS function.

Parameters

- **`(m, n, l) numpy array`** (*`Y`*) – A 3D multi-band image.
- **`Lambda`** (*`float`*) – Regularization parameter #1.
- **`Mu`** (*`float`*) – Regularization parameter #2.
- **`mask`** (*`optional, None, (m, n) numpy array`*) – A sampling mask which is True if the pixel is sampled. Default is None for full sampling.
- **`PCA_transform`** (*`optional, bool`*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **`PCA_th`** (*`optional, int, str`*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.

- **init** (*optional, None, (m, n, l) numpy array*) – The algorithm initialization. Default is None for random initialization.
- **Nit** (*optional, None, int*) – Number of iteration in case of inpainting. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **verbose** (*optional, bool*) – Indicates if information text is desired. Default is True.

Returns

- *(m, n, l) numpy array* – The reconstructed/denoised multi-band image.
- *dict* – A dictionary containing some extra info

Note: Infos in output dictionary:

- `E`: in the case of partial reconstruction, the cost function evolution over iterations.
- `H` : the basis of the chosen signal subspace

References

Monier, E., Oberlin, T., Brun, N., Tencé, M., de Frutos, M., & Dobigeon, N. (2018). Reconstruction of Partially Sampled Multiband Images—Application to STEM-EELS Imaging. IEEE Trans. Comput. Imag., 4(4), 585–598.

3.5.4 inpystem.restore.LS_CLS module

This module implements regularized least square restoration methods adapted to 3D data.

The two methods it gathers are

- **Cosine Least Square (CLS) algorith**,
- **Post-LS Cosine Least Square (Post_LS_CLS) algorithm**.

`inpystem.restore.LS_CLS.`**`CLS`**(*Y*, *Lambda*, *mask=None*, *PCA_transform=True*, *PCA_th='auto'*, *init=None*, *Nit=None*, *verbose=True*)

Cosine Least Square algorithm

The CLS algorithm denoises or reconstructs a multi-band image possibly spatially sub-sampled in the case of spatially sparse content in the DCT basis. It is well adapted to periodic data.

This algorithm solves the folowing optimization problem:

$$\hat{\mathbf{X}} = \underset{\mathbf{X}\in\mathbb{R}^{m\times n\times B}}{\arg\min} \frac{1}{2}||\mathbf{Y}_\mathcal{I} - \mathbf{X}_\mathcal{I}||_\mathrm{F}^2 + \lambda||\mathbf{X}\Psi||_{2,1}$$

where $\mathbf{Y}$ are the corrupted data, $\mathbf{D}$ is a spatial finite difference operator and $\mathcal{I}$ is the set of all sampled pixels.

This algorithm can perform a PCA pre-processing operation to estimate:

- the data subspace basis $\mathbf{H}$,
- the subspace dimension R.

This is particularly usefull to reduce the data dimension and the execution time and to impose a data low-rank property.

> **Caution:** It is strongly recomended to perform PCA before running the algorithm core. This operation is integrated in this function.
>
> In case this pre-processing step has already been done, set the **PCA_transform** parameter to False to disable the PCA step included in the SSS function. If PCA_transform is set to False, the PCA_info parameter is required.

Parameters

- **`(m, n, l) numpy array`** (*`Y`*) – A 3D multi-band image.
- **`Lambda`** (*`float`*) – Regularization parameter.
- **`mask`** (*`optional, None, (m, n) numpy array`*) – A sampling mask which is True if the pixel is sampled. Default is None for full sampling.
- **`PCA_transform`** (*`optional, bool`*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **`PCA_th`** (*`optional, int, str`*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **`init`** (*`optional, None, (m, n, l) numpy array`*) – The algorithm initialization. Default is None for random initialization.
- **`Nit`** (*`optional, None, int`*) – Number of iteration in case of inpainting. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **`verbose`** (*`optional, bool`*) – Indicates if information text is desired. Default is True.

Returns

- *(m, n, l) numpy array* – The reconstructed/denoised multi-band image.
- *dict* – A dictionary containing some extra info

Note: Infos in output dictionary:

- `E` : In the case of partial reconstruction, the cost function evolution over iterations.
- `Gamma` : The array of kept coefficients (order is Fortran-style)
- `nnz_ratio` : the ratio Gamma.size/(m*n)
- `H`: the basis of the chosen signal subspace

`inpystem.restore.LS_CLS.`**`Post_LS_CLS`**(*Y*, *Lambda*, *mask=None*, *PCA_transform=True*, *PCA_th='auto'*, *init=None*, *Nit=None*, *verbose=True*)

Post-Lasso CLS algorithm.

This algorithms consists in applying CLS to restore the data and determine the data support in DCT basis. A post-least square optimization is performed to reduce the coefficients bias.

Parameters

- **`(m, n, l) numpy array`** (*`Y`*) – A 3D multi-band image.

- **Lambda** (*float*) – Regularization parameter.
- **mask** (*optional, None, (m, n) numpy array*) – A sampling mask which is True if the pixel is sampled. Default is None for full sampling.
- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **init** (*optional, None, (m, n, l) numpy array*) – The algorithm initialization. Default is None for random initialization.
- **Nit** (*optional, None, int*) – Number of iteration in case of inpainting. If None, the iterations will stop as soon as the functional no longer evolve. Default is None.
- **verbose** (*optional, bool*) – Indicates if information text is desired. Default is True.

Returns

- *(m, n, l) numpy array* – The reconstructed/denoised multi-band image.
- *tuple* – A 2-tuple whose alements are the CLS and reffitting information dictionaries.

Note: Infos in output dictionary:

- `E_CLS` : In the case of partial reconstruction, the cost function evolution over iterations.
- `E_post_ls` : In the case of partial reconstruction, the cost function evolution over iterations.
- `Gamma` : The array of kept coefficients (order is Fortran-style)
- `nnz_ratio` : the ratio Gamma.size/(m*n)
- `H`: the basis of the chosen signal subspace

3.5.5 inpystem.restore.DL_ITKrMM module

This module implements the ITKrMM algorithm.

The ITKrMM algorithm

`inpystem.restore.DL_ITKrMM.`**`ITKrMM`**(*Y*, *mask=None*, *P=5*, *K=None*, *L=1*, *S=None*, *Nit_lr=10*, *Nit=40*, *init_lr=None*, *init=None*, *CLS_init=None*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)

ITKrMM restoration algorithm.

Parameters

- **Y** (*(m, n) or (m, n, l) numpy array*) – The input data.
- **mask** (*optional, None or (m, n) numpy array*) – The acquisition mask. Default is None for full sampling.
- **P** (*optional, int*) – The width (or height) of the patch. Default is 5.
- **K** (*optional, int*) – The dictionary dimension. Default is 128.
- **L** (*optional, int*) – The number of low rank components to learn. Default is 1.

- **S** (*optional, int*) – The code sparsity level. Default is 20.
- **Nit_lr** (*optional, int*) – The number of iterations for the low rank estimation. Default is 10.
- **Nit** (*optional, int*) – The number of iterations. Default is 40.
- **init** (*(P**2, K+L) or (P**2*l, K+L) numpy array*) – Initialization dictionary.
- **CLS_init** (*optional, dico*) – CLS initialization inofrmation. See Notes for details. Default is None.
- **xref** (*optional, (m, n) or (m, n, l) numpy array*) – Reference image to compute error evolution. Default is None for input Y data.
- **verbose** (*optional, bool*) – The verbose parameter. Default is True.
- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.

Returns

- *(m, n) or (m, n, l) numpy array* – Restored data.
- *dict* – Aditional informations. See Notes.

Notes

The algorithm can be initialized with CLS as soon as `CLS_init` is not None. In this case, `CLS_init` should be a dictionary containing the required `Lambda` key and eventually the `init` optional argument.

The output information keys are:

- `time`: Execution time in seconds.
- `lrc`: low rank component.
- `dico`: Estimated dictionary.
- `E`: Evolution of the error.

The wKSVD algorithm

`inpystem.restore.DL_ITKrMM.`**wKSVD**(*Y, mask=None, P=5, K=None, L=1, S=None, Nit_lr=10, Nit=40, init_lr=None, init=None, CLS_init=None, PCA_transform=True, PCA_th='auto', verbose=True*)

wKSVD restoration algorithm.

Parameters

- **Y** (*(m, n) or (m, n, l) numpy array*) – The input data.
- **mask** (*optional, None or (m, n) numpy array*) – The acquisition mask. Default is None for full sampling.
- **P** (*optional, int*) – The width (or height) of the patch. Default is 5.
- **K** (*optional, int*) – The dictionary dimension. Default is 128.

- **L** (*optional, int*) – The number of low rank components to learn. Default is 1.
- **S** (*optional, int*) – The code sparsity level. Default is 20.
- **Nit_lr** (*optional, int*) – The number of iterations for the low rank estimation. Default is 10.
- **Nit** (*optional, int*) – The number of iterations. Default is 40.
- **init** (*(P**2, K+L) or (P**2*l, K+L) numpy array*) – Initialization dictionary.
- **CLS_init** (*optional, dico*) – CLS initialization inofrmation. See Notes for details. Default is None.
- **xref** (*optional, (m, n) or (m, n, l) numpy array*) – Reference image to compute error evolution. Default is None for input Y data.
- **verbose** (*optional, bool*) – The verbose parameter. Default is True.
- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.

Returns

- *(m, n) or (m, n, l) numpy array* – Restored data.
- *dict* – Aditional informations. See Notes.

Notes

The algorithm can be initialized with CLS as soon as `CLS_init` is not None. In this case, `CLS_init` should be a dictionary containing the required `Lambda` key and eventually the `init` optional argument.

The output information keys are:

- `time`: Execution time in seconds.
- `lrc`: low rank component.
- `dico`: Estimated dictionary.
- `E`: Evolution of the error.

Patch manipulation functions

`inpystem.restore.DL_ITKrMM.`**`forward_patch_transform`**(*ref*, *w*)

Transforms data from 2D/3D array to array whose shape is (w**2, N) where w is the patch width and N is the number of patches.

Parameters

- **ref** (*(m, n) or (m, n, l) numpy array*) – The input image.
- **w** (*int*) – The width (or height) of the patch.

Returns data – The patches stacked version. Its shape is (w**2, N) where N is the number of patches if ref is 2D or (w**2*l, N) is ref is 3D.

Return type (w**2, N) or (w**2*l, N) numpy array inpystem.restore.DL_ITKrMM.**inverse_patch_transform**(*data*, *shape*)

Transforms data from array of the form (w**2, N) or (w**2*l, N) where w is the patch width, l is the number of bands (in the case of 3D data) and N is the number of patches into 2D/3D array.

Parameters

- **data** (*(w**2, N) or (w**2*l, N) numpy array*) – The input data.
- **shape** (*(m, n) or (m, n, l)*) – The image shape.

Returns **ref** – The input image.

Return type (m, n) or (m, n, l) numpy array

CLS initialization function

inpystem.restore.DL_ITKrMM.**CLS_init**(*Y*, *Lambda*, *K=128*, *S=None*, *P=5*, *mask=None*, *PCA_transform=False*, *PCA_th='auto'*, *init=None*, *verbose=True*)

Dictionary learning initialization based on CLS restoration algorithm.

Parameters

- **Y** (*(m, n, l)n umpy array*) – A 3D multi-band image.
- **Lambda** (*float*) – Regularization parameter.
- **K** (*optional, int*) – The dictionary size. Default is 128.
- **S** (*optional, int*) – The code sparsity. Default is 0.1*P*l.
- **P** (*optional, int*) – The patch size. Default is 5.
- **mask** (*optional, None, (m, n) boolean numpy array*) – A sampling mask which is True if the pixel is sampled and False otherwise. Default is None for full sampling.
- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is False as it should be done in dico learning operator.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **init** (*optional, None, (m, n, l) numpy array*) – The algorithm initialization. Default is None for random initialization.
- **verbose** (*optional, bool*) – Indicates if information text is desired. Default is True.

Returns

- *(K, l*P**2) numpy array* – The dictionary for dictionary learning algorithm.
- *(K, l*P**2) numpy array* – The sparse code for dictionary learning algorithm.
- *(m, n, l) numpy array* – CLS restored array.
- *dict* – Dictionary containing some extra info

Note: Infos in output dictionary:

- `E` : In the case of partial reconstruction, the cost function evolution over iterations.
- `Gamma` : The array of kept coefficients (order is Fortran-style)

- `nnz_ratio` : the ratio Gamma.size/(m*n)
- `H`: the basis of the chosen signal subspace

The dictionary learning interface

class `inpystem.restore.DL_ITKrMM.`**`Dico_Learning_Executer`**(*Y*, *mask=None*, *P=5*, *K=None*, *L=1*, *S=None*, *Nit_lr=10*, *Nit=40*, *init_lr=None*, *init=None*, *CLS_init=None*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)

Class to define execute dictionary learning algorithms.

The following class is a common code for most dictionary learning methods. It performs the following tasks:

- reshapes the data in patch format,
- performs low-rank component estimation,
- starts the dictionary learning method,
- reshape output data,
- handle CLS initialization to speed-up computation.

Variables

- **Y** (*(m, n) or (m, n, l) numpy array*) – The input data.
- **Y_PCA** (*(m, n) or (m, n, PCA_th) numpy array*) – The input data in PCA space. Its value is Y if Y is 2D.
- **mask** (*(m, n) numpy array*) – The acquisition mask.
- **P** (*int*) – The width (or height) of the patch.
- **K** (*int*) – The dictionary dimension. This dictionary is composed of L low-rank components and K-L non-low-rank components.
- **L** (*int*) – The number of low rank components to learn.
- **S** (*int*) – The code sparsity level.
- **Nit_lr** (*int*) – The number of iterations for the low rank estimation.
- **Nit** (*int*) – The number of iterations.
- **CLS_init** (*dico*) – CLS initialization inofrmation.
- **verbose** (*bool*) – The verbose parameter. Default is True.
- **mean_std** (*2-tuple*) – Tuple of size 2 which contains the data mean and std.
- **data** (*(N, D) numpy array*) – The Y data in patch format. N (resp. D) is the number of voxels per patch (resp. patches).
- **mdata** (*(N, D) numpy array*) – The mask in patch format. N (resp. D) is the number of voxels per patch (resp. patches).

- **init** (*(N, K-L) numpy array*) – The low-rank estimation initialization in patch format. N is the number of voxels per patch.
- **init** – The dictionary-learning initialization in patch format. N is the number of voxels per patch.
- **PCA_operator** (*PcaHandler object*) – The PCA operator.

Note: The algorithm can be initialized with CLS as soon as `CLS_init` is not None. In this case, `CLS_init` should be a dictionary containing the required `Lambda` key and eventually the CLS `init` optional argument.

The ITKrMM core

`inpystem.restore.DL_ITKrMM.`**rec_lratom**(*data*, *masks=None*, *lrc=None*, *Nit=10*, *inatom=None*, *verbose=True*)

Recover new low rank atom equivalent to itkrmm with K = S = 1.

Parameters

- **data** (*(d, N) numpy array*) – The (corrupted) training signals as its columns.
- **masks** (*(d, N) numpy array*) – Mask data as its columns. masks(.,.) in {0,1}. Default is masks = 1.
- **lrc** (*(d, n) numpy array*) – Orthobasis for already recovered low rank component. Default is None.
- **Nit** (*int*) – Number of iterations. Default is 10.
- **inatom** (*(d,) numpy array*) – Initialisation that should be normalized. Default is None for random.
- **verbose** (*bool*) – If verbose is True, information is sent to the output. Default is True.

Returns **atom** – Estimated low rank component.

Return type (d,) numpy array

`inpystem.restore.DL_ITKrMM.`**OMPm**(*D*, *X*, *S*, *Masks=None*)

Masked OMP.

This is a modified version of OMP to account for corruptions in the signal.

Consider some input data $\mathbf{X}$ (whose shape is (N, P) where N is the number of signals) which are masked by $\mathbf{M}$. Given an input dictionary $\mathbf{D}$ of shape (K, P), this algorithm returns the optimal sparse $\hat{\mathbf{A}}$ matrix such that:

$$\hat{\mathbf{A}} = \arg\min_{\mathbf{A}} \frac{1}{2}||\mathbf{M}\mathbf{X} - \mathbf{M}(\mathbf{A}\mathbf{D})||_F^2$$
$$s.t. \max_k ||\mathbf{A}_{k,:}||_0 \leq S$$

A slightly different modification of Masked OMP is available in "Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing," the book written by M. Elad in 2010.

Parameters

- **D** (*(K, P) numpy array*) – The dictionary. Its rows MUST be normalized, i.e. their norm must be 1.
- **X** (*(N, P) numpy array*) – The masked signals to represent.

- **`S`** (*`int`*) – The max. number of coefficients for each signal.
- **`Masks`** (*`optional, (N, P) numpy array or None`*) – The sampling masks that should be 1 if sampled and 0 otherwise. Default is None for full sampling.

Returns sparse coefficient matrix.

Return type (N, K) sparse coo_matrix array

`inpystem.restore.DL_ITKrMM.`**`itkrmm_core`**(*data*, *masks=None*, *K=None*, *S=1*, *lrc=None*, *Nit=50*, *init=None*, *verbose=True*, *parent=None*)

Iterative Thresholding and K residual Means masked.

Parameters

- **`data`** (*`(d, N) numpy array`*) – The (corrupted) training signals as its columns.
- **`masks`** (*`optional, None, (d, N) numpy array`*) – The masks as its columns. masks(.,.) in {0,1}. Default is None for full sampling.
- **`K`** (*`optional, None or int`*) – Dictionary size. Default is None for d.
- **`S`** (*`optional, int`*) – Desired or estimated sparsity level of the signals. Default is 1.
- **`lrc`** (*`optional, None or (d, L) numpy array`*) – Orthobasis for low rank component. Default is None.
- **`Nit`** (*`optional, int`*) – Number of iterations. Default is 50.
- **`init`** (*`optional, None or (d, K-L) numpy array`*) – Initialisation, unit norm column matrix. Here, L is the number of low rank components. Default is None for random.
- **`verbose`** (*`optional, optional, bool`*) – The verbose parameter. Default is True.
- **`parent`** (*`optional, None or Dico_Learning_Executer object`*) – The Dico_Learning_Executer object that called this function. If this is not None, the SNR between initial true data (given throught the ‘xref‘argument of Dico_Learning_Executer) and the currently reconstructed data will be computed for each iteration. As this means one more OMPm per iteration, this is quite longer. Default is None for faster code and non-SNR output.

Returns

- *(d, K) numpy array* – Estimated dictionary
- *dictionary* – Output information. See Note.

Note: The output dictionary contains the following keys.

- *time* (float): Execution time in seconds.
- ‘SNR’ (None, (Nit,) array): Evolution of the SNR across the iterations in case ‘parent‘is not None.

3.5.6 inpystem.restore.DL_ITKrMM_matlab module

This module implements the ITKrMM algorithm.

The ITKrMM algorithm

`inpystem.restore.DL_ITKrMM_matlab.`**`ITKrMM_matlab`**(*Y*, *mask=None*, *P=5*, *K=None*, *L=1*, *S=None*, *Nit_lr=10*, *Nit=40*, *init_lr=None*, *init=None*, *CLS_init=None*, *save_it=False*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)

ITKrMM restoration algorithm with matlab code.

Parameters

- **`Y`** (*`(m, n) or (m, n, l) numpy array`*) – The input data.
- **`mask`** (*`optional, None, (m, n) numpy array`*) – The acquisition mask. Default is None for full sampling.
- **`P`** (*`optional, int`*) – The width (or height) of the patch. Default is 5.
- **`K`** (*`optional, int`*) – The dictionary dimension. Default is 2*P**2-1.
- **`L`** (*`optional, int`*) – The number of low rank components to learn. Default is 1.
- **`S`** (*`optional, int`*) – The code sparsity level. Default is P-L. This should be lower than K-L.
- **`Nit_lr`** (*`optional, int`*) – The number of iterations for the low rank estimation. Default is 10.
- **`Nit`** (*`optional, int`*) – The number of iterations. Default is 40.
- **`init_lr`** (*`optional, (N, L) numpy array`*) – Initialization for low-rank component. N is the number of voxel in a patch. Default is random initialization.
- **`init`** (*`optional, (N, K-L) numpy array`*) – Initialization for dictionary learning. N is the number of voxel in a patch. Default is random initialization.
- **`CLS_init`** (*`optional, dico`*) – CLS initialization infrmation. See Note for details. Default is None.
- **`save_it`** (*`optional, bool`*) – Particular parameter to save estimated reconstruction inside learning loops. This is recomended to stay false. Default is False.
- **`PCA_transform`** (*`optional, bool`*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **`PCA_th`** (*`optional, int, str`*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **`verbose`** (*`bool`*) – The verbose parameter. Default is True.

Returns

- *(m, n) or (m, n, l) numpy array* – Restored data.
- *dict* – Aditional informations. See Notes.

Notes

The algorithm can be initialized with CLS as soon as `CLS_init` is not None. In this case, `CLS_init` should be a dictionary containing the required `Lambda` key and eventually the CLS `init` optional argument.

The output information keys are:

- `time`: Execution time in seconds.
- `lrc`: low rank component.
- `dico`: Estimated dictionary.
- `E`: Evolution of the error.

The wKSVD algorithm

`inpystem.restore.DL_ITKrMM_matlab.`**`wKSVD_matlab`**(*Y*, *mask=None*, *P=5*, *K=None*, *L=1*, *S=None*, *Nit_lr=10*, *Nit=40*, *init_lr=None*, *init=None*, *CLS_init=None*, *save_it=False*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)

wKSVD restoration algorithm with Matlab code.

Parameters

- **Y** (*(m, n) or (m, n, l) numpy array*) – The input data.
- **mask** (*optional, None, (m, n) numpy array*) – The acquisition mask. Default is None for full sampling.
- **P** (*optional, int*) – The width (or height) of the patch. Default is 5.
- **K** (*optional, int*) – The dictionary dimension. Default is 2*P**2-1.
- **L** (*optional, int*) – The number of low rank components to learn. Default is 1.
- **S** (*optional, int*) – The code sparsity level. Default is P-L. This should be lower than K-L.
- **Nit_lr** (*optional, int*) – The number of iterations for the low rank estimation. Default is 10.
- **Nit** (*optional, int*) – The number of iterations. Default is 40.
- **init_lr** (*optional, (N, L) numpy array*) – Initialization for low-rank component. N is the number of voxel in a patch. Default is random initialization.
- **init** (*optional, (N, K-L) numpy array*) – Initialization for dictionary learning. N is the number of voxel in a patch. Default is random initialization.
- **CLS_init** (*optional, dico*) – CLS initialization infrmation. See Note for details. Default is None.
- **save_it** (*optional, bool*) – Particular parameter to save estimated reconstruction inside learning loops. This is recomended to stay false. Default is False.
- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **verbose** (*bool*) – The verbose parameter. Default is True.

Returns

- *(m, n) or (m, n, l) numpy array* – Restored data.
- *dict* – Aditional informations. See Notes.

Notes

The algorithm can be initialized with CLS as soon as `CLS_init` is not None. In this case, `CLS_init` should be a dictionary containing the required `Lambda` key and eventually the CLS `init` optional argument.

The output information keys are:

- `time`: Execution time in seconds.
- `lrc`: low rank component.
- `dico`: Estimated dictionary.
- `E`: Evolution of the error.

The dictionary learning interface

class `inpystem.restore.DL_ITKrMM_matlab.`**`Matlab_Dico_Learning_Executer`**(*Y, mask=None, P=5, K=None, L=1, S=None, Nit_lr=10, Nit=40, init_lr=None, init=None, CLS_init=None, save_it=False, PCA_transform=True, PCA_th='auto', verbose=True*)

Class to define and execute dictionary learning algorithms with matlab interface.

The following class is a common code for most dictionary learning methods. It performs the following tasks:

- reshapes the data in patch format,
- performs low-rank component estimation,
- starts the dictionary learning method,
- reshape output data,
- handle CLS initialization to speed-up computation.

Variables

- **Y** (*(m, n) or (m, n, l) numpy array*) – The input data.
- **Y_PCA** (*(m, n) or (m, n, PCA_th) numpy array*) – The input data in PCA space. Its value is Y if Y is 2D.
- **mask** (*(m, n) numpy array*) – The acquisition mask.
- **P** (*int*) – The width (or height) of the patch.

- **K** (*int*) – The dictionary dimension. This dictionary is composed of L low-rank components and K-L non-low-rank components.
- **L** (*int*) – The number of low rank components to learn.
- **S** (*int*) – The code sparsity level.
- **Nit_lr** (*int*) – The number of iterations for the low rank estimation.
- **Nit** (*int*) – The number of iterations.
- **CLS_init** (*dico*) – CLS initialization inofrmation.
- **save_it** (*bool*) – Particular parameter to save estimated reconstruction inside learning loops. This is recomended to stay false.
- **verbose** (*bool*) – The verbose parameter. Default is True.
- **mean_std** (*2-tuple*) – Tuple of size 2 which contains the data mean and std.
- **data** (*(N, D) numpy array*) – The Y data in patch format. N (resp. D) is the number of voxels per patch (resp. patches).
- **mdata** (*(N, D) numpy array*) – The mask in patch format. N (resp. D) is the number of voxels per patch (resp. patches).
- **init** (*(N, K-L) numpy array*) – The low-rank estimation initialization in patch format. N is the number of voxels per patch.
- **init** – The dictionary-learning initialization in patch format. N is the number of voxels per patch.
- **PCA_operator** (*PcaHandler object*) – The PCA operator.

Note: The algorithm can be initialized with CLS as soon as `CLS_init` is not None. In this case, `CLS_init` should be a dictionary containing the required `Lambda` key and eventually the CLS `init` optional argument.

3.5.7 inpystem.restore.DL_BPFA module

This module implements the BPFA algorithm.

`inpystem.restore.DL_BPFA.`**`BPFA_matlab`**(*Y*, *mask*, *P=5*, *Omega=1*, *K=128*, *Nit=100*, *step=1*, *PCA_transform=True*, *PCA_th='auto'*, *verbose=True*)

Implements BPFA algorithm for python.

This function does not properly executes BPFA but it calls the Matlab BPFA code.

Parameters

- **Y** (*(m, n) or (m, n, l) numpy array*) – The input data.
- **mask** (*(m, n) numpy array*) – The acquisition mask.
- **P** (*int*) – The patch width. Default is 5.
- **Omega** (*int*) – The Omega parameter. Default is 1.
- **K** (*int*) – The dictionary dimension. Default is 128.
- **Nit** (*int*) – The number of iterations. Default is 100.
- **step** (*int*) – The distance between two consecutive patches. Default is 1 for full overlap.

- **PCA_transform** (*optional, bool*) – Enables the PCA transformation if True, otherwise, no PCA transformation is processed. Default is True.
- **PCA_th** (*optional, int, str*) – The desired data dimension after dimension reduction. Possible values are 'auto' for automatic choice, 'max' for maximum value and an int value for user value. Default is 'auto'.
- **verbose** (*bool*) – The verbose parameter. Default is True.

Returns

- *(m, n) or (m, n, l) numpy array* – Restored data.
- *dict* – Aditional informations. See Notes.

Note: The output information keys are:

- `time`: Execution time in seconds.
- `Z`
- `A`
- `S`

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[APLML+18] Daniele Preziosi, Laura Lopez-Mir, Xiaoyan Li, Tom Cornelissen, Jin Hong Lee, Felix Trier, Karim Bouzehouane, Sergio Valencia, Alexandre Gloter, Agnès Barthélémy, and Manuel Bibes. Direct mapping of phase separation across the metal–insulator transition of ndnio3. *Nano Letters*, 18(4):2226–2232, 2018. doi:10.1021/acs.nanolett.7b04728.

[AZWT+19] Alberto Zobelli, Steffi Y Woo, Luiz HG Tizei, Nathalie Brun, Anna Tararan, Xiaoyan Li, Odile Stéphan, Mathieu Kociak, and Marcel Tencé. Spatial and spectral dynamics in stem hyperspectral imaging using random scan patterns. *arXiv preprint arXiv:1909.07842*, 2019.

[AMonierOberlinBrun+18] É. Monier, T. Oberlin, N. Brun, M. Tencé, M. de Frutos, and N. Dobigeon. Reconstruction of partially sampled multiband images—application to stem-eels imaging. *IEEE Trans. Comput. Imag.*, 4(4):585–598, dec. 2018.

[BNS18] Valeriya Naumova and Karin Schnass. Fast dictionary learning from incomplete data. *EURASIP journal on advances in signal processing*, 2018(1):12, 2018.

[BXZC+12] Z. Xing, M. Zhou, A. Castrodad, G. Sapiro, and L. Carin. Dictionary learning for noisy and incomplete hyperspectral images. *SIAM J. Imag. Sci.*, 5(1):33–56, 2012.

[BMairalEladSapiro08] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *IEEE Trans. Image Process.*, 17(1):53–69, Jan 2008.

[BMonierOberlinBrun+18] É. Monier, T. Oberlin, N. Brun, M. Tencé, M. de Frutos, and N. Dobigeon. Reconstruction of partially sampled multiband images—application to stem-eels imaging. *IEEE Trans. Comput. Imag.*, 4(4):585–598, dec. 2018.

PYTHON MODULE INDEX

i

inpystem, 27
inpystem.dataset, 34
inpystem.dev, 31
inpystem.restore, 44
inpystem.restore.DL_BPFA, 61
inpystem.restore.DL_ITKrMM, 51
inpystem.restore.DL_ITKrMM_matlab, 57
inpystem.restore.interpolation, 45
inpystem.restore.LS_2D, 45
inpystem.restore.LS_3D, 46
inpystem.restore.LS_CLS, 49
inpystem.signals, 27
inpystem.tools, 37
inpystem.tools.dct, 42
inpystem.tools.FISTA, 39
inpystem.tools.matlab_interface, 41
inpystem.tools.metrics, 43
inpystem.tools.misc, 44
inpystem.tools.PCA, 37
inpystem.tools.sec2str, 43

INDEX

Symbols

__init__() (*inpystem.dev.AbstractDev method*), 31
__init__() (*inpystem.dev.Dev2D method*), 32
__init__() (*inpystem.dev.Dev3D method*), 33
__init__() (*inpystem.signals.AbstractStem method*), 29
__init__() (*inpystem.signals.Scan method*), 27
__init__() (*inpystem.tools.FISTA.FISTA method*), 40
__init__() (*inpystem.tools.PCA.PcaHandler method*), 38

A

AbstractDev (*class in inpystem.dev*), 31
AbstractStem (*class in inpystem.signals*), 29
aSAD() (*in module inpystem.tools.metrics*), 43

B

BPFA_matlab() (*in module inpystem.restore.DL_BPFA*), 61

C

CLS() (*in module inpystem.restore.LS_CLS*), 49
CLS_init() (*in module inpystem.restore.DL_ITKrMM*), 54
correct() (*inpystem.signals.AbstractStem method*), 29
correct_fromfile() (*inpystem.signals.AbstractStem method*), 30

D

dct2d() (*in module inpystem.tools.dct*), 42
dct2d_bb() (*in module inpystem.tools.dct*), 42
Dev2D (*class in inpystem.dev*), 32
Dev3D (*class in inpystem.dev*), 33
Dico_Learning_Executer (*class in inpystem.restore.DL_ITKrMM*), 55
Dimension_Reduction() (*in module inpystem.tools.PCA*), 38
direct() (*inpystem.tools.PCA.PcaHandler method*), 38
direct_transform() (*inpystem.dev.Dev3D method*), 34

E

EigenEstimate() (*in module inpystem.tools.PCA*), 39
execute() (*inpystem.tools.FISTA.FISTA method*), 41

F

FISTA (*class in inpystem.tools.FISTA*), 40
forward_patch_transform() (*in module inpystem.restore.DL_ITKrMM*), 53
from_file() (*inpystem.signals.Scan class method*), 28

G

get_mask() (*inpystem.signals.Scan method*), 28

I

idct2d() (*in module inpystem.tools.dct*), 42
idct2d_bb() (*in module inpystem.tools.dct*), 42
inpystem (*module*), 27
inpystem.dataset (*module*), 34
inpystem.dev (*module*), 31
inpystem.restore (*module*), 44
inpystem.restore.DL_BPFA (*module*), 61
inpystem.restore.DL_ITKrMM (*module*), 51
inpystem.restore.DL_ITKrMM_matlab (*module*), 57
inpystem.restore.interpolation (*module*), 45
inpystem.restore.LS_2D (*module*), 45
inpystem.restore.LS_3D (*module*), 46
inpystem.restore.LS_CLS (*module*), 49
inpystem.signals (*module*), 27
inpystem.tools (*module*), 37
inpystem.tools.dct (*module*), 42
inpystem.tools.FISTA (*module*), 39
inpystem.tools.matlab_interface (*module*), 41
inpystem.tools.metrics (*module*), 43
inpystem.tools.misc (*module*), 44
inpystem.tools.PCA (*module*), 37
inpystem.tools.sec2str (*module*), 43

`interpolate()` (*in module inpystem.restore.interpolation*), 45
`inverse()` (*inpystem.tools.PCA.PcaHandler method*), 38
`inverse_patch_transform()` (*in module inpystem.restore.DL_ITKrMM*), 54
`inverse_transform()` (*inpystem.dev.Dev3D method*), 34
`ITKrMM()` (*in module inpystem.restore.DL_ITKrMM*), 51
`itkrmm_core()` (*in module inpystem.restore.DL_ITKrMM*), 57
`ITKrMM_matlab()` (*in module inpystem.restore.DL_ITKrMM_matlab*), 58

L

`L1_LS()` (*in module inpystem.restore.LS_2D*), 45
`load_file()` (*in module inpystem.dataset*), 35
`load_key()` (*in module inpystem.dataset*), 36

M

`Matlab_Dico_Learning_Executer` (*class in inpystem.restore.DL_ITKrMM_matlab*), 60
`matlab_interface()` (*in module inpystem.tools.matlab_interface*), 41

N

`NMSE()` (*in module inpystem.tools.metrics*), 43

O

`OMPm()` (*in module inpystem.restore.DL_ITKrMM*), 56

P

`PcaHandler` (*class in inpystem.tools.PCA*), 37
`plot()` (*inpystem.dev.Dev2D method*), 32
`plot()` (*inpystem.signals.AbstractStem method*), 30
`plot()` (*inpystem.signals.Scan method*), 29
`plot_as1D()` (*inpystem.dev.Dev3D method*), 34
`plot_as1D()` (*inpystem.signals.Stem3D method*), 30
`plot_as2D()` (*inpystem.dev.Dev3D method*), 34
`plot_as2D()` (*inpystem.signals.Stem3D method*), 30
`plot_roi()` (*inpystem.dev.Dev3D method*), 34
`plot_roi()` (*inpystem.signals.Stem3D method*), 30
`plot_sum()` (*inpystem.signals.Stem3D method*), 30
`Post_LS_CLS()` (*in module inpystem.restore.LS_CLS*), 50

R

`random()` (*inpystem.signals.Scan class method*), 28
`ratio()` (*inpystem.signals.Scan property*), 28
`read_data_path()` (*in module inpystem.dataset*), 35
`rec_lratom()` (*in module inpystem.restore.DL_ITKrMM*), 56
`restore()` (*inpystem.dev.Dev2D method*), 32
`restore()` (*inpystem.dev.Dev3D method*), 34
`restore()` (*inpystem.signals.AbstractStem method*), 30
`restore()` (*inpystem.signals.Stem2D method*), 30
`restore()` (*inpystem.signals.Stem3D method*), 30

S

`Scan` (*class in inpystem.signals*), 27
`sec2str()` (*in module inpystem.tools.sec2str*), 43
`seed()` (*inpystem.dev.AbstractDev property*), 31
`set_data_path()` (*in module inpystem.dataset*), 35
`set_ndata()` (*inpystem.dev.AbstractDev method*), 31
`show_sum()` (*inpystem.dev.Dev3D method*), 34
`sigma()` (*inpystem.dev.AbstractDev property*), 31
`SNN()` (*in module inpystem.restore.LS_3D*), 48
`SNR()` (*in module inpystem.tools.metrics*), 43
`SSIM()` (*in module inpystem.tools.metrics*), 44
`SSS()` (*in module inpystem.restore.LS_3D*), 46
`Stem2D` (*class in inpystem.signals*), 30
`Stem3D` (*class in inpystem.signals*), 30
`StopCritera()` (*inpystem.tools.FISTA.FISTA method*), 40
`StopTest()` (*inpystem.tools.FISTA.FISTA method*), 41

T

`toslice()` (*in module inpystem.tools.misc*), 44

W

`wKSVD()` (*in module inpystem.restore.DL_ITKrMM*), 52
`wKSVD_matlab()` (*in module inpystem.restore.DL_ITKrMM_matlab*), 59