Toupy Documentation

Release 0.1.0

Julio C. da Silva

Jul 27, 2021

INTRODUCTION

CHAPTER

ONE

WELCOME TO TOUPY DOCUMENTATION

Toupy is a suite of tools for the processing of high-resolution tomography dataset compiled by J. C. da Silva and licensed under the GPLv3 license .

The name **toupy** stands for **Tomographic Utilites for Python** and it is a wordplay with the French word *toupie* for spinning top, the toy designed to spin rapidly on the ground, the motion of which causes it to remain precisely balanced on its tip due to its rotational inertia. Here you can find the wikipedia page in English and in French.

Toupy implements tools for preprocessing the projections before the tomographic reconstruction and the reconstruction itself. The pipeline and algorithms is briefly summarized in this publication[1], but which is also based on previous works[2].

1.1 Quicklinks

- Get started quickly with the examples in the *templates* directory.
- The complete documentation.
- The source code can be obtained via Github repo of toupy .

1.2 References

1.3 Acknowledgments

- Ana Diaz, PSI, Switzerland
- Andreas Menzel, PSI, Switzerland
- Manuel Guizar-Sicairos, PSI, Switzerland
- Pierre Paleo, ESRF, France
- Peter Cloetens, ESRF, France

[1] da Silva, J. C., Haubrich, J., Requena, G., Hubert, M., Pacureanu, A., Bloch, L., Yang, Y., Cloetens, P., *High energy near-and far-field ptychographic tomography at the ESRF*. **Proc. SPIE** 10391, Developments in X-Ray Tomography XI, 1039106 (2017). doi

[2] Guizar-Sicairos, M., Diaz, A., Holler, M., Lucas, M. S., Menzel, A., Wepf, R. A., and Bunk, O., *Phase tomography from x-ray coherent diffractive imaging projections*, **Opt. Express** 19, 21345–21357 (2011). doi

CHAPTER

TWO

INSTALLATION

2.1 Requirements

The required packages are list in the file requirements.txt. Before the installation of Toupy, you should run:

```
pip install -r requirements.txt
```

2.2 Via pip install

```
pip install toupy
```

or for a local installation, using the flag –user:

```
pip install --user toupy
```

2.3 Via Python Package

Clone it first from git:

```
git clone https://github.com/jcesardasilva/toupy.git
```

change into toupy directory:

```
cd toupy
```

The installation should be as simple as:

```
sudo python3 setup.py install
```

or, for local installation, using the flag –user:

```
python3 setup.py install --user
```

CHAPTER

THREE

TEMPLATES

The templates here are routines to analyze the X-ray tomographic data. It is exemplified by the case of data acquired at ID16A beamline of ESRF, but it can easily adapted to datas from any other beamline.

They consists basically into an step of load the projections data, the main step to be adapted for different beamlines, the implementation of processing step and, finally the saving step. For ID16A data, you should not need to adapt the code.

3.1 Tomographic Reconstruction

The tomographic reconstruction of high resolution data is divided in several steps:

- Processing of the projections, for example, the phase-retrieval for phase contrast imaging, the fluorescence fitting for XRF-tomographic dataset and other.
- Restoration of the projections in case of phase wrapping or linear phase ramp for phase-contrast imaging, and the normalization of the XRF datasets.
- Alignment of the stack of projections.
- Finally, the tomographic reconstruction.

The templates here will then guide you through the steps above. After each step, the files are saved and can be used for the next step. All the files, except Tiff conversion, are saved in HDF5 format. For the analysis, it is important to have the latest version of Python packages. For people working with data from ID16A and with access to beamline computing resources, this can be obtained by using the ID16A Python environment, which is activated by typing *py3venv_on* on the Linux prompt.

The python scripts can be run from shell, from ipython or from Jupyter notebook. This dependes on the user`s preference. For illustration purposes only, the description below supposes you will launch the scripts from shell.

3.1.1 Loading of the projections

Edit *load_projections.py* with proper parameters and run

```
python load_projections.py
```

The instructions of what to do appear on the screen. It loads either .tif or .edf files. The next step consists of the vertical registration of the projections.

3.1.2 Linear Phase ramp removal

Edit *remove_phase_ramp.py* with proper parameters and run

```
python remove_phase_ramp.py
```

This open a GUI interface with buttons to allow to proceed with the phase ramp removal.

3.1.3 Phase unwrapping

Edit *phase_unwrapping.py* with proper parameters and run

```
python phase_unwrapping.py
```

The instructions of what to do appear on the screen.

3.1.4 Vertical alignment

Edit *vertical_alignment.py* with proper parameters and run

```
python vertical_alignment.py
```

The instructions of what to do appear on the screen.

3.1.5 Derivatives of the Projection

Edit *projections_derivatives.py* with proper parameters and run

```
python projections_derivatives.py
```

The instructions of what to do appear on the screen.

3.1.6 Sinogram inspection

Edit *sinogram_inspection.py* with proper parameters and run

```
python sinogram_inspection.py
```

The instructions of what to do appear on the screen.

3.1.7 Horizontal alignment

Edit *horizontal_alignment.py* with proper parameters and run

```
python horizontal_alignment.py
```

The instructions of what to do appear on the screen.

3.1.8 Tomographic reconstruction

Edit *tomographic_reconstruction.py* with proper parameters and run

```
python tomographic_reconstruction.py
```

The instructions of what to do appear on the screen.

3.1.9 Tiff 8 or 16 bits conversion

This step is only necessary for people who want to have the tomographic slices as tiff rather than as HDF5.

Edit *tiff_conversion.py* with proper parameters and run

```
python tiff_conversion.py
```

The instructions of what to do appear on the screen.

CHAPTER

FOUR

TOUPY.IO PACKAGE

4.1 Submodules

4.2 toupy.io.dataio module

class toupy.io.dataio.**LoadData**(***params*)

Bases: *toupy.io.dataio.PathName*, toupy.io.dataio.Variables

Load projections from HDF5 file

classmethod load(**args*, ***params*)

Load data from h5 file

Parameters

- **h5name** (*str*) – File name from which data is loaded
- ****params** – Dictionary of additonal parameters
- **params["autosave"]** (*bool*) – Save the projections once load without asking
- **params["phaseonly"]** (*bool*) – Load only phase projections. Used when the projections are complex-valued.
- **params["amponly"]** (*bool*) – Load only amplitude projections. Used when the projections are complex-valued.
- **params["pixtol"]** (*float*) – Tolerance for alignment, which is also used as a search step
- **params["alignx"]** (*bool*) – True or False to activate align x using center of mass (default= False, which means align y only)
- **params["shiftmeth"]** (*str*) – Shift images with fourier method (default). The options are *linear* -> Shift images with linear interpolation (default); *fourier* -> Fourier shift or *spline* -> Shift images with spline interpolation.
- **params["circle"]** (*bool*) – Use a circular mask to eliminate corners of the tomogram
- **params["filtertype"]** (*str*) – Filter to use for FBP
- **params["freqcutoff"]** (*float*) – Frequency cutoff for tomography filter (between 0 and 1)
- **params["cliplow"]** (*float*) – Minimum value in tomogram
- **params["cliphigh"]** (*float*) – Maximum value in tomogram
- **params["correct_bad"]** (*bool*) – If true, it will interpolate bad projections. The numbers of projections to be corrected is given by *params["bad_projs"]*.

- **params["bad_projs"]** (*list of ints*) – List of projections to be interpolated. It starts at 0.

Returns

- **stack_projs** (*array_like*) – Stack of projections
- **theta** (*array_like*) – Stack of thetas
- **shiftstack** (*array_like*) – Shifts in vertical (1st dimension) and horizontal (2nd dimension)
- **datakwargs** (*dict*) – Dictionary with metadata information

classmethod load_olddata(**args*, ***params*)

Load old data from h5 file. It should disappear soon.

Parameters

- **h5name** (*str*) – File name from which data is loaded
- ****params** – Dictionary of additonal parameters
- **params["autosave"]** (*bool*) – Save the projections once load without asking
- **params["phaseonly"]** (*bool*) – Load only phase projections. Used when the projections are complex-valued.
- **params["amponly"]** (*bool*) – Load only amplitude projections. Used when the projections are complex-valued.
- **params["pixtol"]** (*float*) – Tolerance for alignment, which is also used as a search step
- **params["alignx"]** (*bool*) – True or False to activate align x using center of mass (default= False, which means align y only)
- **params["shiftmeth"]** (*str*) – Shift images with fourier method (default). The options are *linear* -> Shift images with linear interpolation (default); *fourier* -> Fourier shift or *spline* -> Shift images with spline interpolation.
- **params["circle"]** (*bool*) – Use a circular mask to eliminate corners of the tomogram
- **params["filtertype"]** (*str*) – Filter to use for FBP
- **params["freqcutoff"]** (*float*) – Frequency cutoff for tomography filter (between 0 and 1)
- **params["cliplow"]** (*float*) – Minimum value in tomogram
- **params["cliphigh"]** (*float*) – Maximum value in tomogram
- **params["correct_bad"]** (*bool*) – If true, it will interpolate bad projections. The numbers of projections to be corrected is given by *params["bad_projs"]*.
- **params["bad_projs"]** (*list of ints*) – List of projections to be interpolated. It starts at 0.

Returns

- **stack_projs** (*array_like*) – Stack of projections
- **theta** (*array_like*) – Stack of thetas
- **shiftstack** (*array_like*) – Shifts in vertical (1st dimension) and horizontal (2nd dimension)
- **datakwargs** (*dict*) – Dictionary with metadata information

classmethod loadmasks(**args*, ***params*)

Load masks from previous h5 file

Parameters **h5name** (*str*) – File name from which data is loaded

Returns **masks** – Array with the masks

Return type array_like

classmethod loadshiftstack(**args*, ***params*)
Load shitstack from previous h5 file

Parameters **h5name** (*str*) – File name from which data is loaded

Returns **shiftstack** – Shifts in vertical (1st dimension) and horizontal (2nd dimension)

Return type array_like

classmethod loadtheta(**args*, ***params*)
Load shitstack from previous h5 file

Parameters **h5name** (*str*) – File name from which data is loaded

Returns **shiftstack** – Shifts in vertical (1st dimension) and horizontal (2nd dimension)

Return type array_like

class toupy.io.dataio.**LoadProjections**(***params*)
Bases: *toupy.io.dataio.PathName*, toupy.io.dataio.Variables

Load the reconstructed projections from the ptyr files

check_angles()
Find the angles of the projections and plot them to be checked Specific to ID16A beamline (ESRF)

check_angles_new()
Find the angles of the projections and plot them to be checked Specific to ID16A beamline (ESRF)

static insert_missing(*stack_objs*, *theta*, *missingnum*)
Insert missing projections by interpolation of neighbours

classmethod load(***params*)
Load the reconstructed projections from phase-retrieved files.

Parameters

- ****params** – Container with parameters to load the files.
- **params["account"]** (*str*) – User experiment number at ESRF.
- **params["samplename"]** (*str*) – Sample name
- **params["pathfilename"]** (*str*) – Path to the first projection file.
- **params["regime"]** (*str*) – Imaging regime. The options are: *nearfield*, *farfield*, *holoct*.
- **params["showrecons"]** (*bool*) – To show or not the projections once loaded
- **params["autosave"]** (*bool*) – Save the projections once load without asking
- **params["phaseonly"]** (*bool*) – Load only phase projections. Used when the projections are complex-valued.
- **params["amponly"]** (*bool*) – Load only amplitude projections. Used when the projections are complex-valued.
- **params["border_crop_x"]** (*int, None*) – Amount of pixels to crop at each border in x.
- **params["border_crop_y"]** (*int, None*) – Amount of pixels to crop at each border in y.
- **params["checkextraprojs"]** (*bool*) – Check for the projections acquired at and over 180 degrees.

- **params["missingprojs"]** (*bool*) – Allow to interpolate for missing projections. The numbers of the projections need to be provided in params["missingnum"].
- **params["missingnum"]** (*list of ints*) – Numbers of the missing projections to be interpolated.

Returns

- **stack_objs** (*array_like*) – Array containing the projections
- **stack_angles** (*array_like*) – Array containing the thetas
- **pxsize** (*list of floats*) – List containing the pixel size in the vertical and horizontal directions. Typically, the resolution is isotropic and the two values are the same
- **paramsload** (*dict*) – Parameters of the loading

`classmethod loadedf(**params)`
Load the reconstructed projections from the edf files This is adapted for the phase-contrast imaging generating projections as edf files

Parameters

- ****params** – Container with parameters to load the files.
- **params["account"]** (*str*) – User experiment number at ESRF.
- **params["samplename"]** (*str*) – Sample name
- **params["pathfilename"]** (*str*) – Path to the first projection file.
- **params["regime"]** (*str*) – Imaging regime. The options are: *nearfield*, *farfield*, *holoct*.
- **params["showrecons"]** (*bool*) – To show or not the projections once loaded
- **params["autosave"]** (*bool*) – Save the projections once load without asking

Returns

- **stack_objs** (*array_like*) – Array containing the projections
- **stack_angles** (*array_like*) – Array containing the thetas
- **pxsize** (*list of floats*) – List containing the pixel size in the vertical and horizontal directions. Typically, the resolution is isotropic and the two values are the same
- **paramsload** (*dict*) – Parameters of the loading

`class toupy.io.dataio.LoadTomogram(**params)`
Bases: *toupy.io.dataio.LoadData*

Load projections from HDF5 file

`classmethod load(*args, **params)`
Load tomographic data from h5 file

Parameters **args[0]** (*str*) – HDF5 file name from which data is loaded

Returns

- **tomogram** (*array_like*) – Stack of tomographic slices
- **theta** (*array_like*) – Stack of thetas
- **shiftstack** (*array_like*) – Shifts in vertical (1st dimension) and horizontal (2nd dimension)
- **datakwargs** (*dict*) – Dictionary with metadata information

`class toupy.io.dataio.PathName(**params)`
Bases: `object`

Class to manage file location and paths

datafilewcard()
Create file wildcard to search for files

metadatafilewcard()
Create file wildcard to search for metafiles

results_datapath(*h5name*)
create path for the h5file in result folder

results_folder()
create path for the result folder

search_projections()
Search for projection given the filenames

class toupy.io.dataio.**SaveData**(***params*)
Bases: *toupy.io.dataio.PathName*, toupy.io.dataio.Variables

Save projections to HDF5 file

classmethod **save**(**args*, ***params*)
Save data to HDF5 File

Parameters

- ***args** – positional arguments
- **args[0]** (*str*) – H5 file name
- **args[1]** (*array_like*) – Array containing the stack of projections
- **args[2]** (*array_like*) – Values of theta
- **args[3]** (*array_like*) – Array containing the shifts for each projection in the stack. If not provided, it will be initialized with zeros
- **args[4]** (*array_like or None*) – Array containing the projection masks

classmethod **saveFSC**(**args*, ***params*)
Save FSC data to HDF5 file

Parameters

- ***args** – positional arguments
- **args[0]** (*str*) – H5 file name
- **args[1]** (*array_like*) – Normalized frequencies
- **args[2]** (*array_like*) – Value of the threshold for each frequency
- **args[3]** (*array_like*) – The FSC curve
- **args[4]** (*array_like*) – The first tomogram
- **args[5]** (*array_like*) – The second tomogram
- **args[6]** (*array_like*) – The array of theta values
- **args[7]** (*float*) – Pixel size

savecheck()
Decorator for save data

classmethod **savemasks**(**args*, ***params*)

class toupy.io.dataio.**SaveTomogram**(***params*)
Bases: *toupy.io.dataio.SaveData*

Save tomogram to HDF5 file

classmethod **convert_to_tiff**(**args*, ***params*)
Convert the HDF5 file with the tomogram to tiff

Parameters

- ***args** – positional arguments
- **args[0]** (*str*) – H5 file name
- **args[1]** (*array_like*) – Array containing the stack of slices (tomogram)
- **args[2]** (*array_like*) – Values of theta
- **args[3]** (*array_like*) – Array containing the shifts for each projection in the stack

classmethod **save**(**args*, ***params*)

Parameters

- ***args** – positional arguments
- **args[0]** (*str*) – H5 file name
- **args[1]** (*array_like*) – Array containing the stack of slices (tomogram)
- **args[2]** (*array_like*) – Values of theta
- **args[3]** (*array_like*) – Array containing the shifts for each projection in the stack

classmethod **save_vol_to_h5**(**args*, ***params*)

savecheck()
Decorator for save data

tiff_folderpath(*foldername*)
Create the path to the folder in which the tiff files will be stored.

toupy.io.dataio.**remove_extraprojs**(*stack_projs*, *theta*)
Remove extra projections of tomographic scans with projections at 180, 90 and 0 degrees at the end

Parameters

- **stack_projs** (*array_like*) – Stack of projections with the first index correspoding to the projection number
- **theta** (*array_like*) – Array of theta values

Returns

- **stack_projs** (*array_like*) – Stack of projections after the removal
- **theta** (*array_like*) – Array of theta values after the removal

4.3 toupy.io.filesrw module

Files read and write

toupy.io.filesrw.**convert16bitstiff**(*tiffimage*, *low_cutoff*, *high_cutoff*)
Convert 16 bits tiff files back to quantitative values.

Parameters

- **imgpath** (*array_like*) – Image read from 16 bits tiff file.
- **low_cutoff** (*float*) – Low cutoff of the gray level.
- **high_cutoff** (*float*) – High cutoff of the gray level.

Returns **tiffimage** – Array containing the image with quantitative values.

Return type array_like

`toupy.io.filesrw.convert8bitstiff(filename, low_cutoff, high_cutoff)`
Convert 8bits tiff files back to quantitative values.

Parameters

- **imgpath** (*array_like*) – Image read from 8 bits tiff file.
- **low_cutoff** (*float*) – Low cutoff of the gray level.
- **high_cutoff** (*float*) – High cutoff of the gray level.

Returns **tiffimage** – Array containing the image with quantitative values.

Return type array_like

`toupy.io.filesrw.convertimageto16bits(input_image, low_cutoff, high_cutoff)`
Convert image gray-level to 16 bits with normalization

Parameters

- **input_image** (*array_like*) – Input image to be converted.
- **low_cutoff** (*float*) – Low cutoff of the gray level.
- **high_cutoff** (*float*) – High cutoff of the gray level.

Returns **tiffimage** – Array containing the image at 16 bits.

Return type array_like

`toupy.io.filesrw.convertimageto8bits(input_image, low_cutoff, high_cutoff)`
Convert image gray-level to 8 bits with normalization.

Parameters

- **input_image** (*array_like*) – Input image to be converted.
- **low_cutoff** (*float*) – Low cutoff of the gray level.
- **high_cutoff** (*float*) – High cutoff of the gray level.

Returns **tiffimage** – Array containing the image at 8 bits.

Return type array_like

`toupy.io.filesrw.create_paramsh5(**params)`
Create parameter file in HDF5 format

Parameters **params** (*dict*) – Dictionary containing the parameters to be saved

`toupy.io.filesrw.crop_array(input_array, delcropx, delcropy)`
Crop borders from 2D arrays

Parameters

- **input_array** (*array_like*) – Input array to be cropped
- **delcropx** (*int*) – Number of pixels to be crop from borders in x and y directions
- **delcropy** (*int*) – Number of pixels to be crop from borders in x and y directions

Returns **cropped_array** – Cropped array

Return type array_like

`toupy.io.filesrw.load_paramsh5(**params)`
Load parameters from HDF5 file of parameters

`toupy.io.filesrw.memmap_volfile(filename)`
Memory map the tomogram from .vol file

Parameters **filename** (*str*) – filename to be read

Returns

- **tomogram** (*array_like*) – 3D array containing the tomogram
- **voxelSize** (*floats*) – Voxel size in meters
- **arrayshape** (*tuple of floats*) – The array shape: (x_size, y_size, z_size)

Examples

```
>>> volpath = 'volfilename.vol'
>>> tomogram,voxelsize,arrayshape = memmap_volfile(volpath)
```

Note: The volume info file containing the metadata of the volume should be in the same folder as the volume file.

`toupy.io.filesrw.`**`read_cxi`**(*pathfilename*, *correct_orientation=True*)
Read reconstruction files .cxi from PyNX

Parameters

- **`pathfilename`** (`str`) – Path to file
- **`correct_orientation`** (`bool`) – True for correcting the image orientation and False to keep as it is. The default value is `True`.

Returns

- **data1** (*array_like, complex*) – Object image
- **probe1** (*array_like, complex*) – Probe images
- **pixelsize** (*list of floats*) – List with pixelsizes in vertical and horizontal directions
- **energy** (*float*) – Energy of the incident photons

Examples

```
>>> imgpath = 'filename.cxi'
>>> objdata, probedata, pixel, energy = read_cxi(imgpath)
```

`toupy.io.filesrw.`**`read_edf`**(*fname*)
Read EDF files of tomographic datasets

Parameters **`fname`** (`str`) – Path to file

Returns

- **projs** (*array_like*) – Array of projections
- **pixelsize** (*list of floats*) – List with pixelsizes in vertical and horizontal directions
- **energy** (*float*) – Energy of the incident photons
- **nvue** (*int*) – Number of projections

`toupy.io.filesrw.`**`read_ptyr`**(*pathfilename*, *correct_orientation=True*)
Read reconstruction files .ptyr from Ptypy

Parameters

- **`pathfilename`** (`str`) – Path to file
- **`correct_orientation`** (`bool, optional`) – True for correcting the image orientation and False to keep as it is. The default value is `True`.

Returns

- **data1** (*array_like, complex*) – Object image
- **probe1** (*array_like, complex*) – Probe images
- **pixelsize** (*list of floats*) – List with pixelsizes in vertical and horizontal directions
- **energy** (*float*) – Energy of the incident photons

Examples

```
>>> imgpath = 'filename.ptyr'
>>> objdata, probedata, pixel, energy = read_ptyr(imgpath)
```

toupy.io.filesrw.**read_recon**(*filename*, *correct_orientation=False*)
Wrapper for choosing the function to read recon file

Parameters

- **pathfilename** (*str*) – Path to file
- **correct_orientation** (*bool, optional*) – True for correcting the image orientation and False to keep as it is. The default value is False.

Returns

- **data1** (*array_like, complex*) – Object image
- **probe1** (*array_like, complex*) – Probe images
- **pixelsize** (*list of floats*) – List with pixelsizes in vertical and horizontal directions
- **energy** (*float*) – Energy of the incident photons

Examples

```
>>> imgpath = 'filename.ptyr'
>>> objdata, probedata, pixel, energy = read_recon(imgpath)
```

toupy.io.filesrw.**read_theta_raw**(*pathfilename*)
Auxiliary function to read theta from raw data acquired at ID16A

Parameters **pathfilename** (*str*) – Path to file

Returns **theta** – Tomographic angle

Return type float

Examples

```
>>> imgpath = 'filename.h5'
>>> theta = read_theta_raw(imgpath)
```

toupy.io.filesrw.**read_theta_recon**(*reconfile*)
Auxiliary function to read theta from recon files

Parameters **reconfile** (*str*) – Path to recon file

Returns **theta** – Tomographic angle

Return type float

Examples

```
>>> imgpath = 'filename.ptyr'
>>> theta = read_theta_recon(imgpath)
```

toupy.io.filesrw.**read_tiff**(*imgpath*)

Read tiff files using skimage.io.imread

Parameters **imgpath** (*str*) – Path to tiff file with extension

Returns **imgout** – Array containing the image

Return type array_like

Examples

```
>>> imgpath = 'image.tiff'
>>> ar = read_tiff(imgpath)
>>> ar.dtype
dtype('uint16')
>>> np.max(ar)
65535
```

toupy.io.filesrw.**read_tiff_info**(*tiff_info_file*)

Read info file from tiff slices of the reconstructed tomographic volume

Parameters **tiff_info_file** (*str*) – Info filename

Returns

- **low_cutoff** (*float*) – Low cutoff of the gray level
- **high_cutoff** (*float*) – High cutoff of the gray level
- **pixelsize** (*float*) – Pixelsize in nanometers

Note: The info file here is the file that is save when the volume is exported to Tiff files. It is not the info file saved by the volume reconstruction when saving the file in .vol.

toupy.io.filesrw.**read_volfile**(*filename*)

Read tomogram from .vol file

Parameters **filename** (*str*) – filename to be read

Returns

- **tomogram** (*array_like*) – 3D array containing the tomogram
- **voxelsize** (*floats*) – Voxel size in meters
- **arrayshape** (*tuple of floats*) – The array shape: (x_size, y_size, z_size)

Examples

```
>>> volpath = 'volfilename.vol'
>>> tomogram,voxelsize,arrayshape = read_volfile(volpath)
```

Note: The volume info file containing the metadata of the volume should be in the same folder as the volume file.

`toupy.io.filesrw.`**`write_edf`**(*fname*, *data_array*, *hd=None*)
Write EDF files

Parameters

- **fname** (*str*) – File name
- **data_array** (*array_like*) – Data to be saved as edf
- **hd** (*dict*) – Dictionary with header information

`toupy.io.filesrw.`**`write_paramsh5`**(*h5filename*, ***params*)
Writes params to HDF5 file

Parameters

- **h5filename** (*str*) – Filename of the params file
- **params** (*dict*) – Dictionary containing the parameters to be saved

`toupy.io.filesrw.`**`write_tiff`**(*input_array*, *pathfilename*, *plugin='tifffile'*)
Write tiff files using skimag.io.imsave

Parameters

- **input_array** (*array_like*) – Input array to be saved
- **pathfilename** (*str*) – Path and filename to save the file

`toupy.io.filesrw.`**`write_tiffmetadata`**(*filename*, *low_cutoff*, *high_cutoff*, *factor*, ***params*)
Creates a txt file with the information about the Tiff normalization

Parameters

- **filename** (*str*) – Filename to save the file.
- **low_cutoff** (*float*) – Low cutoff value for the tiff normalization.
- **high_cutoff** (*float*) – High cutoff value for the tiff normalization.
- **factor** (*float*) – Multiplicative factor in case it is needed.
- **params** (*dict*) – Dictionary of additional parameters.
- **params["voxelsize"]** (*float*) – Voxel size.
- **params["filtertype"]** (*str*) – Filter used in the tomographic reconstruction.
- **params["freqcutoff"]** (*float*) – Frequency cutoff used in the tomographic reconstruction.
- **params["bits"]** (*int*) – The tiff type. Options: *8* for 8 bits or *16* for 16 bits.

4.4 toupy.io.h5chunk_shape_3D module

toupy.io.h5chunk_shape_3D.**__all__ = ['binlist', 'numVals', 'perturbShape', 'chunk_shape_3D']**

toupy.io.h5chunk_shape_3D.**binlist**(*n*, *width=0*)
Return list of bits that represent a non-negative integer.

Parameters

- **n** (*int*) – non-negative integer
- **width** (*int*) – number of bits in returned zero-filled list (default 0)

toupy.io.h5chunk_shape_3D.**chunk_shape_3D**(*varShape*, *valSize=4*, *chunkSize=4096*)
Return a 'good shape' for a 3D variable, assuming balanced 1D/(n-1)D access[1]

Parameters

- **varShape** (*sequence of ints*) – length 3 list of variable dimension sizes
- **chunkSize** (*int, optional*) – maximum chunksize desired, in bytes (default 4096)
- **valSize** (*int, optional*) – size of each data value, in bytes (default 4)

Returns Returns integer chunk lengths of a chunk shape that provides balanced access of 1D subsets and 2D subsets of a netCDF or HDF5 variable var with shape (T, X, Y), where the 1D subsets are of the form var[:,x,y] and the 2D slices are of the form var[t,:,:], typically 1D time series and 2D spatial slices.

Return type tuple

Notes

'Good shape' for chunks means that the number of chunks accessed to read either kind of 1D or 2D subset is approximately equal, and the size of each chunk (uncompressed) is no more than chunkSize, which is often a disk block size. Code fetched from[2] and[3].

References

toupy.io.h5chunk_shape_3D.**numVals**(*shape*)
Return number of values in chunk of specified shape, given by a list of dimension lengths.

Parameters **shape** (*sequence of ints*) – list of variable dimension sizes

toupy.io.h5chunk_shape_3D.**perturbShape**(*shape*, *onbits*)
Return shape perturbed by adding 1 to elements corresponding to 1 bits in onbits

Parameters

- **shape** (*sequence of ints*) – list of variable dimension sizes
- **onbits** (*int*) – non-negative integer less than 2**len(shape)

[1] https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking_data_choosing_shapes
[2] https://www.unidata.ucar.edu/blog_content/data/2013/chunk_shape_3D.py
[3] https://github.com/HDFGroup/datacontainer/blob/master/lib/chunking.py

CHAPTER

FIVE

TOUPY.REGISTRATION PACKAGE

5.1 Submodules

5.2 toupy.registration.registration module

toupy.registration.registration.**alignprojections_horizontal**(*sinogram*, *theta*, *shiftstack*, ***params*)

Function to align projections by tomographic consistency[1],[2]. It relies on having already aligned the vertical direction. The code aligns using the consistency before and after tomographic combination of projections.

Parameters

- **sinogram** (*array_like*) – Sinogram derivative, the second index should be the angle
- **theta** (*array_like*) – Reconstruction angles (in degrees). Default: m angles evenly spaced between 0 and 180 (if the shape of *radon_image* is (N, M)).
- **shiftstack** (*array_like*) – Array with initial estimates of positions
- **params** (*dict*) – Container with parameters for the registration
- **params["pixtol"]** (*float*) – Tolerance for alignment, which is also used as a search step
- **params["alignx"]** (*bool*) – True or False to activate align x using center of mass (default= False, which means align y only)
- **params["shiftmeth"]** (*str*) – Shift images with fourier method (default). The options are *linear* -> Shift images with linear interpolation (default); *fourier* -> Fourier shift or *spline* -> Shift images with spline interpolation.
- **params["circle"]** (*bool*) – Use a circular mask to eliminate corners of the tomogram
- **params["filtertype"]** (*str*) – Filter to use for FBP
- **params["freqcutoff"]** (*float*) – Frequency cutoff for tomography filter (between 0 and 1)
- **params["cliplow"]** (*float*) – Minimum value in tomogram
- **params["cliphigh"]** (*float*) – Maximum value in tomogram

Returns

- **shiftstack** (*array_like*) – Corrected object positions
- **alinedsinogram** (*array_like*) – Array containting the aligned sinogram

[1] Guizar-Sicairos, M., et al., "Quantitative interior x-ray nanotomography by a hybrid imaging technique," Optica 2, 259-266 (2015).
[2] da Silva, J. C., et al., "High energy near-and far-field ptychographic tomography at the ESRF," Proc. SPIE 10391, Developments in X-Ray Tomography XI, 1039106 (2017).

References

toupy.registration.registration.**alignprojections_vertical**(*input_stack*, *shiftstack*, ***params*)

Vertical alignment of projections using mass fluctuation approach[3],[4]. It relies on having air on both sides of the sample (non local tomography). It performs a local search in y, so convergence issues can be addressed by giving an approximate initial guess for a possible drift via shiftstack

Parameters

- **input_stack** (*array_like*) – Stack of projections
- **limrow** (*list of ints*) – Limits of window of interest in y
- **limcol** (*list of ints*) – Limits of window of interest in x
- **shiftstack** (*array_like*) – Array of initial estimates for object motion (2,n)
- **params** (*dict*) – Container with parameters for the registration
- **params['pixtol']** (*float*) – Tolerance for alignment, which is also used as a search step
- **params['polyorder']** (*int*) – Specify the polynomial order of bias removal. For example: polyorder = 1 -> mean, polyorder = 2 -> linear).
- **params['alignx']** (*bool*) – True or False to activate align x using center of mass (default= False, which means align y only)
- **params['shiftmeth']** (*str*) – Shift images with fourier method (default). The options are *linear* -> Shift images with linear interpolation (default); *fourier* -> Fourier shift or *spline* -> Shift images with spline interpolation.

Returns

- **shiftstack** (*array_like*) – Corrected bject positions
- **input_stack** (*array_like*) – Aligned stack of the projections

References

toupy.registration.registration.**center_of_mass_stack**(*input_stack*, *lims*, *shiftstack*, *shift_method='fourier'*)

Calculates the center of the mass for each projection in the stack and returns a stack of centers of mass (row, col) i.e., returns shiftstack[1] If the array is zero, it return the center of mass at 0.

toupy.registration.registration.**compute_aligned_horizontal**(*input_stack*, *shiftstack*, *shift_method='linear'*)

Compute the alignment of the stack on at the horizontal direction

Parameters

- **input_array** (*array_like*) – Stack of images to be shifted
- **shiftstack** (*array_like*) – Array of initial estimates for object motion (2,n) The estimates for vertical movement will be changed to 0
- **shift_method** (*str (default linear)*) – Name of the shift method. Options: 'linear', 'fourier', 'spline'

Returns **output_stack** – 2D function containing the stack of aligned images

Return type array_like

[3] Guizar-Sicairos, M., et al. , "Phase tomography from x-ray coherent diffractive imaging projections," Opt. Express 19, 21345-21357 (2011).

[4] da Silva, J. C., et al. "High energy near-and far-field ptychographic tomography at the ESRF," Proc. SPIE 10391, Developments in X-Ray Tomography XI, 1039106 (2017)

`toupy.registration.registration.`**`compute_aligned_sino`**(*input_sino*, *shiftslice*, *shift_method='linear'*)

Compute the aligned sinogram given the correction for object positions

Parameters

- **`input_sino`** (*`array_like`*) – Input sinogram to be shifted
- **`shiftslice`** (*`array_like`*) – Array of estimates for object motion (1,n)
- **`shift_method`** (*`str (default linear)`*) – Name of the shift method. Options: 'linear', 'fourier', 'spline'

Returns **output_sino** – 2D function containing the aligned sinogram

Return type array_like

`toupy.registration.registration.`**`compute_aligned_stack`**(*input_stack*, *shiftstack*, *shift_method='linear'*)

Compute the aligned stack given the correction for object positions

Parameters

- **`input_array`** (*`array_like`*) – Stack of images to be shifted
- **`shiftstack`** (*`array_like`*) – Array of initial estimates for object motion (2,n)
- **`shift_method`** (*`str (default linear)`*) – Name of the shift method. Options: 'linear', 'fourier', 'spline'

Returns **output_stack** – 2D function containing the stack of aligned images

Return type array_like

`toupy.registration.registration.`**`estimate_rot_axis`**(*input_array*, *theta*, ***params*)

Initial estimate of the rotation axis

`toupy.registration.registration.`**`oneslicefordisplay`**(*sinogram*, *theta*, ***params*)

Calculate one slice for display.

Parameters

- **`sinogram`** (*`array_like`*) – Sinogram derivative, the second index should be the angle
- **`theta`** (*`array_like`*) – Reconstruction angles (in degrees). Default: m angles evenly spaced between 0 and 180 (if the shape of *radon_image* is (N, M)).
- **`params`** (*`dict`*) – Container with parameters for the registration.
- **`params["filtertype"]`** (*`str`*) – Filter to use for FBP
- **`params["freqcutoff"]`** (*`float`*) – Frequency cutoff for tomography filter (between 0 and 1)

`toupy.registration.registration.`**`refine_horizontalalignment`**(*input_stack*, *theta*, *shiftstack*, ***params*)

Refine horizontal alignment. Please, see the description of each parameter in *`alignprojections_horizontal()`*.

`toupy.registration.registration.`**`register_2Darrays`**(*image1*, *image2*)

Image registration. Register two images using phase cross correlations.

Parameters

- **`image1`** (*`array_like`*) – Image of reference
- **`image2`** (*`array_like`*) – Image to be shifted relative to image1

Returns

- **shift** (*list of floats*) – List of shifts applied, with the row shift in the 1st dimension and the column shift in the 2nd dimension.

- **diffphase** (*float*) – Difference of phase between the two images
- **offset_image2** (*array_like*) – Shifted image2 relative to image1

`toupy.registration.registration.`**`tomoconsistency_multiple`**(*input_stack*, *theta*, *shiftstack*, ***params*)

Apply tomographic consistency alignement on multiple slices. By default is implemented over 10 slices.

Parameters

- **Input_stack** (*array_like*) – Stack of projections
- **theta** (*array_like*) – Reconstruction angles (in degrees). Default: m angles evenly spaced between 0 and 180 (if the shape of *radon_image* is (N, M)).
- **shiftstack** (*array_like*) – Array with initial estimates of positions
- **params** (*dict*) – Dictionary with additional parameters for the alignment. Please, see the description of each parameter in `alignprojections_horizontal()`.

Returns **shiftstack** – Average of the object shifts over 10 slices

Return type array_like

`toupy.registration.registration.`**`vertical_fluctuations`**(*input_stack*, *lims*, *shiftstack*, *shift_method='fourier'*, *polyorder=2*)

Calculate the vertical fluctuation functions of a stack

Parameters

- **input_array** (*array_like*) – Stack of images to be shifted
- **lims** (*list of ints*) – Limits of rows and columns to be considered. lims=[limrow,limcol]
- **shiftstack** (*array_like*) – Array of initial estimates for object motion (2,n)
- **shift_method** (*str, optional*) – Name of the shift method. Options: 'linear', 'fourier', 'spline'. The default method is 'linear'.
- **polyorder** (*int, optional*) – Order of the polynomial to remove bias from the mass fluctuation function. The default value is 2.

Returns **vert_fluct** – 2D function containing the mass fluctuation after shift and bias removal for the stack of images

Return type array_like

`toupy.registration.registration.`**`vertical_shift`**(*input_array*, *lims*, *vstep*, *maxshift*, *shift_method='linear'*, *polyorder=2*)

Calculate the vertical shift of an array

Parameters

- **input_array** (*array_like*) – Image to be shifted
- **lims** (*list of ints*) – Limits of rows and columns to be considered. lims=[limrow,limcol]
- **vstep** (*float*) – Amount to shift the input_array vertically
- **maxshift** (*float*) – Maximum value of the shifts in order to avoid border problems
- **shift_method** (*str, optional*) – Name of the shift method. Options: 'linear', 'fourier', 'spline'. The default method is 'linear'.
- **polyorder** (*int, optional*) – Order of the polynomial to remove bias from the mass fluctuation function. The default value is 2.

Returns **shift_cal** – 1D function containing the mass fluctuation after shift and bias removal

Return type array_like

5.3 toupy.registration.shift module

class toupy.registration.shift.**ShiftFunc**(***params*)

Bases: toupy.registration.shift.Variables

Collections of shift fuctions

__call__(**args*)

Implement the shifts

Parameters ***args** –

args[0] [array_like] Input array

args[1] [int or tuple] Shift amplitude

args[2] [str (optional)] Padding mode if necessary

args[3] [bool (optional)] True for complex output or False for real output

shift_fft(*input_array*, *shift*)

Performs pixel and subpixel shift (with wraping) using pyFFTW.

Since FFTW has efficient functions for array sizes which can be decompose in prime factor, the input_array is padded to the next fast size given by pyFFTW.next_fast_len. The padding is done in mode = 'reflect' by default to reduce border artifacts.

Parameters

- **input_array** (*array_like*) – Input image to calculate the shifts.
- **shift** (*int or tuple*) – Number of pixels to shift. For 1D, use a integer value. For 2D, use a tuple of integers where the first value corresponds to shifts in the rows and the second value corresponds to shifts in the columns.

Returns **output_array** – Shifted image

Return type array_like

shift_linear(*input_array*, *shift*)

Shifts an image with wrap around and bilinear interpolation

Parameters

- **input_array** (*array_like*) – Input image to calculate the shifts.
- **shift** (*int or tuple*) – Number of pixels to shift. For 1D, use a integer value. For 2D, use a tuple of integers where the first value corresponds to shifts in the rows and the second value corresponds to shifts in the columns.

Returns **output_array** – Shifted image

Return type array_like

shift_spline_wrap(*input_array*, *shift*)

Performs pixel and subpixel shift (with wraping) using splines

Parameters

- **input_array** (*array_like*) – Input image to calculate the shifts.
- **shift** (*int or tuple*) – Number of pixels to shift. For 1D, use a integer value. For 2D, use a tuple of integers where the first value corresponds to shifts in the rows and the second value corresponds to shifts in the columns.

Returns **output_array** – Shifted image

Return type array_like

CHAPTER SIX

TOUPY.RESOLUTION PACKAGE

6.1 Submodules

6.2 toupy.resolution.FSC module

FOURIER SHELL CORRELATION modules

class toupy.resolution.FSC.**FSCPlot**(*img1*, *img2*, *threshold='halfbit'*, *ring_thick=1*, *apod_width=20*)

Bases: *toupy.resolution.FSC.FourierShellCorr*

Upper level object to plot the FSC and threshold curves

Parameters

- **img1** (*ndarray*) – A 2-dimensional array containing the first image
- **img2** (*ndarray*) – A 2-dimensional array containing the second image
- **threshold** (*str, optional*) – The option *onebit* means 1 bit threshold with `SNRt = 0.5`, which should be used for two independent measurements. The option *halfbit* means 1/2 bit threshold with `SNRt = 0.2071`, which should be use for split tomogram. The default option is `half-bit`.
- **ring_thick** (*int, optional*) – Thickness of the frequency rings. Normally the pixels get assined to the closest integer pixel ring in Fourier Domain. With ring_thick, each ring gets more pixels and more statistics. The default value is `1`.
- **apod_width** (*int, optional*) – Width in pixel of the edges apodization. It applies a Hanning window of the size of the data to the data before the Fourier transform calculations to attenuate the border effects. The default value is `20`.

Returns

- **fn** (*ndarray*) – A 1-dimensional array containing the frequencies normalized by the Nyquist frequency
- **FSC** (*ndarray*) – A 1-dimensional array containing the Fourier Shell correlation curve
- **T** (*ndarray*) – A 1-dimensional array containing the threshold curve

plot()

class toupy.resolution.FSC.**FourierShellCorr**(*img1*, *img2*, *threshold='halfbit'*, *ring_thick=1*, *apod_width=20*)

Bases: `object`

Computes the Fourier Shell Correlation[1] between image1 and image2, and estimate the resolution based on the threshold funcion T of 1 or 1/2 bit.

Parameters

[1] M. van Heel, M. Schatzb, *Fourier shell correlation threshold criteria,* Journal of Structural Biology 151, 250-262 (2005)

- **img1** (*ndarray*) – A 2-dimensional array containing the first image
- **img2** (*ndarray*) – A 2-dimensional array containing the second image
- **threshold** (*str, optional*) – The option *onebit* means 1 bit threshold with `SNRt = 0.5`, which should be used for two independent measurements. The option *halfbit* means 1/2 bit threshold with `SNRt = 0.2071`, which should be use for split tomogram. The default option is `half-bit`.
- **ring_thick** (*int, optional*) – Thickness of the frequency rings. Normally the pixels get assined to the closest integer pixel ring in Fourier Domain. With ring_thick, each ring gets more pixels and more statistics. The default value is `1`.
- **apod_width** (*int, optional*) – Width in pixel of the edges apodization. It applies a Hanning window of the size of the data to the data before the Fourier transform calculations to attenuate the border effects. The default value is `20`.

Returns

- **FSC** (*ndarray*) – Fourier Shell correlation curve
- **T** (*ndarray*) – Threshold curve

Note: If 3D images, the first axis is the number of slices, ie., `[slices, rows, cols]`

References

apodization()
Compute the Hanning window of the size of the data for the apodization

Note: This method does not depend on the parameter `apod_width` from the class

circle()
Create circle with apodized edges

fouriercorr()
Method to compute FSC and threshold

nyquist()
Evaluate the Nyquist Frequency

ringthickness()
Define indexes for ring_thick

transverse_apodization()
Compute a tapered Hanning-like window of the size of the data for the apodization

6.3 toupy.resolution.FSCtools module

FOURIER SHELL CORRELATION

toupy.resolution.FSCtools.**compute_2tomograms**(*sinogram*, *theta*, ***params*)
Split the tomographic dataset in 2 datasets and compute 2 tomograms from them.

Parameters

- **sinogram** (*ndarray*) – A 2-dimensional array containing the sinogram
- **theta** (*ndarray*) – A 1-dimensional array of thetas

Returns

- **recon1** (*ndarray*) – A 2-dimensional array containing the 1st reconstruction
- *recon2* – A 2-dimensional array containing the 2nd reconstruction

`toupy.resolution.FSCtools.`**`compute_2tomograms_splitted`**(*sinogram1*, *sinogram2*, *theta1*, *theta2*, ***params*)

Compute 2 tomograms from already splitted tomographic dataset

Parameters

- **sinogram1** (*ndarray*) – A 2-dimensional array containing the sinogram 1
- **sinogram2** (*ndarray*) – A 2-dimensional array containing the sinogram 2
- **theta1** (*ndarray*) – A 1-dimensional array of thetas for sinogram1
- **theta2** (*ndarray*) – A 1-dimensional array of thetas for sinogram2

Returns

- **recon1** (*ndarray*) – A 2-dimensional array containing the 1st reconstruction
- *recon2* – A 2-dimensional array containing the 2nd reconstruction

`toupy.resolution.FSCtools.`**`split_dataset`**(*sinogram*, *theta*)

Split the tomographic dataset in 2 datasets

Parameters

- **sinogram** (*ndarray*) – A 2-dimensional array containing the sinogram
- **theta** (*ndarray*) – A 1-dimensional array of thetas

Returns

- **sinogram1** (*ndarray*) – A 2-dimensional array containing the 1st sinogram
- *sinogram2* – A 2-dimensional array containing the 2nd sinogram
- **theta1** (*ndarray*) – A 1-dimensional array containing the 1st set of thetas
- **theta2** (*ndarray*) – A 1-dimensional array containing the 2nd set of thetas

CHAPTER

SEVEN

TOUPY.RESTORATION PACKAGE

7.1 Submodules

7.2 toupy.restoration.GUI_tracker module

class toupy.restoration.GUI_tracker.**AmpTracker**(*fig*, *ax1*, *ax2*, *X1*, ***params*)

Bases: *toupy.restoration.GUI_tracker.PhaseTracker*

Widgets for the phase ramp removal

Note: It inherits most of the functionality of *PhaseTracker*, except the ones related to amplitude projections rather than to phase projections. Please, refert to the docstring of *PhaseTracker* for further description.

apply_all_masks(*event*)

Apply the linear air correction using current mask and log to all projections

apply_mask(*event*)

Apply the air correction using current mask and apply the log

class toupy.restoration.GUI_tracker.**PhaseTracker**(*fig*, *ax1*, *ax2*, *X1*, ***params*)

Bases: object

Widgets for the phase ramp removal

add_mask(*event*)

Add the mask to the plot

apply_all_masks(*event*)

Apply the linear phase correction using current mask to all projections

apply_mask(*event*)

Apply the linear phase correction using current mask

cmvmax(*val*)

Set the vmax equals to `val` on colormap

cmvmin(*val*)

Set the vmin equals to `val` on colormap

down(*event*)

Move projection number down using button Prev

draw_mask(*event*)

Draw the mask using roipoly

key_event(*event*)

Move projection number up/down using right/left arrows in the keyboard

load_masks(*event*)
Load masks from file

mask_all(*event*)
Use the same mask for all projections

onclose(*event*)
Close the figure

onscroll(*event*)
Move projection number up/down using the mouse scroll wheel

play(*event*)
Plot one project after the other (play)

remove_all_mask(*event*)
Remove all the masks

remove_mask(*event*)
Remove the current selected area from the mask

remove_ramp(*event*)
Remove linear phase ramp

remove_rampall(*event*)
Remove linear phase ramp of all

save_masks(*event*)
Save mask to file

submit(*text*)
Textbox submit

unwrapping_all(*event*)
Unwrap phase of all projections

unwrapping_phase(*event*)
Unwrap phase

up(*event*)
Move projection number up using button Next

update()
Update the plot canvas

toupy.restoration.GUI_tracker.**gui_plotamp**(*stack_objs*, ***params*)
GUI for the air removal from amplitude projections

Parameters

- **stack_objs** (*array_like*) – Stack of amplitude projections
- **params** (*dict*) – Dictionary of additonal parameters
- **params["autosave"]** (*bool*) – Save the projections once load without asking
- **params["correct_bad"]** (*bool*) – If true, it will interpolate bad projections. The numbers of projections to be corrected is given by *params["bad_projs"]*.
- **params["bad_projs"]** (*list of ints*) – List of projections to be interpolated. It starts at 0.
- **params["vmin"]** (*float, None*) – Minimum value of gray-level to display
- **params["vmax"]** (*float, None*) – Maximum value of gray-level to display

Returns **stack_ampcorr** – Stack of corrected amplitude projections

Return type array_like

toupy.restoration.GUI_tracker.**gui_plotphase**(*stack_objs*, ***params*)

GUI for the phase ramp removal from phase projections

Parameters

- **stack_objs** (*array_like*) – Stack of phase projections
- **params** (*dict*) – Dictionary of additonal parameters
- **params["autosave"]** (*bool*) – Save the projections once load without asking
- **params["correct_bad"]** (*bool*) – If true, it will interpolate bad projections. The numbers of projections to be corrected is given by *params["bad_projs"]*.
- **params["bad_projs"]** (*list of ints*) – List of projections to be interpolated. It starts at 0.
- **params["vmin"]** (*float, None*) – Minimum value of gray-level to display
- **params["vmax"]** (*float, None*) – Maximum value of gray-level to display

Returns **stack_phasecorr** – Stack of corrected phase projections

Return type array_like

7.3 toupy.restoration.derivativetools module

toupy.restoration.derivativetools.**calculate_derivatives**(*stack_array*, *roiy*, *roix*, *shift_method='fourier'*)

Compute projection derivatives

Parameters

- **stack_array** (*array_like*) – Input stack of arrays to calculate the derivatives
- **roix** (*tuple*) – Limits of the area on which to calculate the derivatives
- **roiy** (*tuple*) – Limits of the area on which to calculate the derivatives
- **shift_method** (*str*) – Name of the shift method to use. For the available options, please see `ShiftFunc()` in *toupy.registration*

Returns **aligned_diff** – Stack of derivatives of the arrays along the horizontal direction

Return type array_like

toupy.restoration.derivativetools.**calculate_derivatives_fft**(*stack_array*, *roiy*, *roix*, *n_cpus=-1*)

Compute projection derivatives using FFTs

Parameters

- **stack_array** (*array_like*) – Input stack of arrays to calculate the derivatives
- **roix** (*tuple*) – Limits of the area on which to calculate the derivatives
- **roiy** (*tuple*) – Limits of the area on which to calculate the derivatives
- **n_cpus** (*int*) – The number of cpus for parallel computing. If *n_cpus<0*, the number of cpus will be determined by *os.cpu_counts()*

Returns **aligned_diff** – Stack of derivatives of the arrays along the horizontal direction

Return type array_like

toupy.restoration.derivativetools.**chooseregiontoderivatives**(*stack_array*, ***params*)

Choose the region to be unwrapped

toupy.restoration.derivativetools.**derivatives**(*input_array*, *shift_method='fourier'*)

Calculate the derivative of an image

Parameters

- **input_array** (*array_like*) – Input image to calculate the derivatives
- **shift_method** (*str*) – Name of the shift method to use. For the available options, please see ShiftFunc() in *toupy.registration*

Returns **diffimg** – Derivatives of the images along the row direction

Return type array_like

toupy.restoration.derivativetools.**derivatives_fft**(*input_img*, *symmetric=True*, *n_cpus=- 1*)
Calculate the derivative of an image using FFT along the horizontal direction

Parameters

- **input_array** (*array_like*) – Input image to calculate the derivatives
- **symmetric** (*bool*) – If *True*, symmetric difference is calculated
- **n_cpus** (*int*) – The number of cpus for parallel computing. If *n_cpus<0*, the number of cpus will be determined by *os.cpu_counts()*

Returns **diffimg** – Derivatives of the images along the row direction

Return type array_like

toupy.restoration.derivativetools.**derivatives_sino**(*input_sino*, *shift_method='fourier'*)
Calculate the derivative of the sinogram

Parameters

- **input_array** (*array_like*) – Input sinogram to calculate the derivatives
- **shift_method** (*str*) – Name of the shift method to use. For the available options, please see ShiftFunc() in *toupy.registration*

Returns **diffsino** – Derivatives of the sinogram along the radial direction

Return type array_like

toupy.restoration.derivativetools.**gradient_axis**(*x*, *axis=- 1*)
Compute the gradient (keeping dimensions) along one dimension only. By default, the axis is -1 (diff along columns).

7.4 toupy.restoration.ramptools module

toupy.restoration.ramptools.**rmair**(*image*, *mask*)
Correcting amplitude factor using the mask from the phase ramp removal considering only pixels where mask is unity, arrays have center on center of array

Parameters

- **image** (*array_like*) – Amplitude-contrast image
- **mask** (*bool*) – Boolean array with indicating the locations from where the air value should be obtained

Returns **normalizedimage** – Image normalized by the air values

Return type array_like

toupy.restoration.ramptools.**rmlinearphase**(*image*, *mask*)
Removes linear phase from object

Parameters

- **image** (*array_like*) – Input image
- **mask** (*bool*) – Boolean array with ones where the linear phase should be computed from

Returns **im_output** – Linear ramp corrected image

Return type array_like

`toupy.restoration.ramptools.`**`rmphaseramp`**(*a*, *weight=None*, *return_phaseramp=False*)

Auxiliary functions to attempt to remove the phase ramp in a two-dimensional complex array `a`.

Parameters

- **a** (*array_like*) – Input image as complex 2D-array.
- **weight** (*array_like, str, optional*) – Pass weighting array or use `'abs'` for a modulus-weighted phaseramp and `Non` for no weights.
- **return_phaseramp** (*bool, optional*) – Use True to get also the phaseramp array `p`.

Returns

- **out** (*array_like*) – Modified 2D-array, `out=a*p`
- **p** (*array_like, optional*) – Phaseramp if `return_phaseramp = True`, otherwise omitted

Note: Function forked from Ptypy.plot_utils (https://github.com/ptycho/ptypy) and ported to Python 3.

Examples

```
>>> b = rmphaseramp(image)
>>> b, p = rmphaseramp(image , return_phaseramp=True)
```

7.5 toupy.restoration.roipoly module

Draw polygon regions of interest (ROIs) in matplotlib images, similar to Matlab's roipoly function. See the file example.py for an application. Created by Joerg Doepfert 2014 based on code posted by Daniel Kornhauser.

class `toupy.restoration.roipoly.`**`roipoly`**(*fig=[]*, *ax=[]*, *roicolor='b'*)

Bases: `object`

`displayMean`(*currentImage*, ***textkwargs*)

`displayROI`(***linekwargs*)

`getMask`(*currentImage*)

7.6 toupy.restoration.unwraptools module

`toupy.restoration.unwraptools.`**`chooseregiontounwrap`**(*stack_array*, *threshold=5000*, *parallel=False*, *ncores=1*)

Choose the region to be unwrapped

Parameters

- **stack_array** (*ndarray*) – A 3-dimensional array containing the stack of projections to be unwrapped.
- **threshold** (*int, optional*) – The threshold of the number of acceptable phase residues. (Default = 5000)
- **parallel** (*bool, optional*) – If *True*, multiprocessing and threading will be used. (Default = *False*)

Returns

- **rx, ry** (*tuple*) – Limits of the area to be unwrapped
- **airpix** (*tuple*) – Position of the pixel which should contains only air/vacuum

`toupy.restoration.unwraptools.`**`distance`**`(`*pixel1*, *pixel2*`)`

Return the Euclidean distance of two pixels.

Example

```
>>> distance(np.arange(1,10),np.arange(2,11))
3.0
```

`toupy.restoration.unwraptools.`**`get_charge`**`(`*residues*`)`

Get the residues charges

Parameters **`residues`** (*`ndarray`*) – A 2-dimensional array containing the with residues

Returns

- **posres** (*array_like*) – Positions of the residues with positive charge
- **negres** (*array_like*) – Positions of the residues with negative charge

`toupy.restoration.unwraptools.`**`phaseresidues`**`(`*phimage*`)`

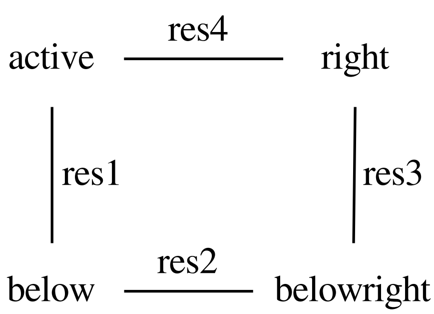
Calculates the phase residues[1] for a given wrapped phase image.

Parameters **`phimage`** (*`ndarray`*) – A 2-dimensional array containing the phase-contrast images with gray-level in radians

Returns **residues** – A 2-dimensional array containing the map of residues (valued +1 or -1)

Return type ndarray

Note: Note that by convention the positions of the phase residues are marked on the top left corner of the 2 by 2 regions as shown below:



Inspired by PhaseResidues.m created by B.S. Spottiswoode on 07/10/2004 and by find_residues.m created by Manuel Guizar - Sept 27, 2011

---

[1] R. M. Goldstein, H. A. Zebker and C. L. Werner, Radio Science 23, 713-720 (1988).

References

toupy.restoration.unwraptools.**phaseresiduesStack**(*stack_array*, *threshold=5000*)

Calculate the map of residues on the stack

Parameters **stack_array** (*ndarray*) – A 3-dimensional array containing the stack of projections from which to calculate the phase residues.

Returns

- **resmap** (*array_like*) – Phase residue map
- **posres** (*tuple*) – Positions of the residues in the format `posres = (yres,xres)`

toupy.restoration.unwraptools.**phaseresiduesStack_parallel**(*stack_array*, *threshold=1000*, *ncores=2*)

Calculate the map of residues on the stack

Parameters

- **stack_array** (*ndarray*) – A 3-dimensional array containing the stack of projections from which to calculate the phase residues.
- **threshold** (*int, optional*) – The threshold of the number of acceptable phase residues. (Default = 5000)

Returns

- **resmap** (*array_like*) – Phase residue map
- **posres** (*tuple*) – Positions of the residues in the format `posres = (yres,xres)`

toupy.restoration.unwraptools.**unwrapping_phase**(*stack_phasecorr*, *rx*, *ry*, *airpix*, ***params*)

Unwrap the phase of the projections in a stack.

Parameters

- **stack_phasecorr** (*ndarray*) – A 3-dimensional array containing the stack of projections to be unwrapped
- **rx** (*tuple or list of ints*) – Limits of the are to be unwrapped in x and y
- **ry** (*tuple or list of ints*) – Limits of the are to be unwrapped in x and y
- **airpix** (*tuple or list of ints*) – Position of pixel in the air/vacuum area
- **params** (*dict*) – Dictionary of additional parameters
- **params["vmin"]** (*float, None*) – Minimum value for the gray level at each display
- **params["vmin"]** – Maximum value for the gray level at each display

Returns **stack_unwrap** – A 3-dimensional array containing the stack of unwrapped projections

Return type ndarray

Note: It uses the phase unwrapping algorithm by Herraez et al.[2] implemented in Scikit-Image (https://scikit-image.org).

[2] Miguel Arevallilo Herraez, David R. Burton, Michael J. Lalor, and Munther A. Gdeisat, "Fast two-dimensional phase-unwrapping algorithm based on sorting by reliability following a noncontinuous path", Journal Applied Optics, Vol. 41, No. 35, pp. 7437, 2002

References

`toupy.restoration.unwraptools.wrap`(*phase*)

Wrap a scalar value or an entire array to [-0.5, 0.5).

Parameters **phase** (*float or array_like*) – The value or signal to wrapped.

Returns Wrapped value or array

Return type float or array

Note: Created by Sebastian Theilenberg, PyMRR, which is available at Github repository: https://github.com/theilen/PyMRR.git

`toupy.restoration.unwraptools.wraptopi`(*phase*, *endpoint=True*)

Wrap a scalar value or an entire array

Parameters

- **phase** (*float or array_like*) – The value or signal to wrapped.
- **endpoint** (*bool, optional*) – If `endpoint=False`, the scalar value or array is wrapped to [-pi, pi), whereas if `endpoint=True`, it is wrapped to (-pi, pi]. The default value is `endpoint=True`

Returns Wrapped value or array

Return type float or array

Example

```
>>> import numpy as np
>>> wraptopi(np.linspace(-np.pi,np.pi,7),endpoint=True)
array([ 3.14159265, -2.0943951 , -1.04719755, -0.        ,  1.04719755,
    2.0943951 ,  3.14159265])
>>> wraptopi(np.linspace(-np.pi,np.pi,7),endpoint=False)
array([-3.14159265, -2.0943951 , -1.04719755,  0.        ,  1.04719755,
    2.0943951 , -3.14159265])
```

7.7 toupy.restoration.vorticestools module

`toupy.restoration.vorticestools.cart2pol`(*x*, *y*)

Change from cartesian to polar coordinates

Parameters

- **x** (*array_like*) – Values in cartesian coordinates
- **y** (*array_like*) – Values in cartesian coordinates

Returns **rho, phi** – Values in polar coordinates

Return type array_like

`toupy.restoration.vorticestools.get_object_novort`(*img_phase*, *residues*)

Remove the vortices from the phase projections

Parameters

- **img_phase** (*array_like*) – Phase image with vortices to be removed without linear phase ramp

- **residues** (*array_like*) – Residues map

Returns

- **img_phase_novort** (*array_like*) – Phase image without vortices
- **xres, yres** (*array_like*) – Coordinates *x* and *y* of the vortices

`toupy.restoration.vorticestools.`**`get_probe_novort`**`(`*img_phase*, *residues*`)`

Remove the vortices from the probe

Parameters

- **img_phase** (*array_like*) – Probe image with vortices to be removed without linear phase ramp
- **residues** (*array_like*) – Residues map

Returns

- **img_phase_novort** (*array_like*) – Probe image without vortices
- **xres, yres** (*array_like*) – Coordinates *x* and *y* of the vortices

`toupy.restoration.vorticestools.`**`pol2cart`**`(`*rho*, *phi*`)`

Change from polar to cartesian coordinates

Parameters

- **rho** (*array_like*) – Values in polar coordinates
- **phi** (*array_like*) – Values in polar coordinates

Returns **x, y** – Values in cartesian coordinates

Return type array_like

`toupy.restoration.vorticestools.`**`rmvortices_object`**`(`*img_in*, *to_ignore=100*`)`

Remove phase vortices on the object image ignoring an amount of pixels equals to `to_ignore` from the borders.

Parameters

- **img_phase** (*array_like*) – Phase image with vortices to be removed.
- **to_ignore** (*int, optional*) – amount of pixels to ignore from the borders.

Returns

- **img_phase_novort** (*array_like*) – Phase image without vortices
- **xres, yres** (*array_like*) – Coordinates *x* and *y* of the vortices

Note: An eventual linear phase ramp will be remove from the input image.

`toupy.restoration.vorticestools.`**`rmvortices_probe`**`(`*img_in*, *to_ignore=100*`)`

Remove phase vortices on the probe image ignoring an amount of pixels equals to `to_ignore` from the borders.

Parameters

- **img_phase** (*array_like*) – Probe image with vortices to be removed.
- **to_ignore** (*int, optional*) – amount of pixels to ignore from the borders.

Returns

- **img_phase_novort** (*array_like*) – Probe image without vortices
- **xres, yres** (*array_like*) – Coordinates *x* and *y* of the vortices

Note: An eventual linear phase ramp will be remove from the input image.

CHAPTER

EIGHT

TOUPY.SIMULATION PACKAGE

8.1 Submodules

8.2 toupy.simulation.phantom_creator module

Module to create the Shepp-Logan phantom for simulation Forked from https://jenda.hrach.eu/f2/cat-py/phantom.py

`toupy.simulation.phantom_creator.`**`phantom`**(*N=256*, *phantom_type='Modified Shepp-Logan'*, *ellipses=None*)

Create a Shepp-Logan[1] or modified Shepp-Logan phantom[2] . A phantom is a known object (either real or purely mathematical) that is used for testing image reconstruction algorithms. The Shepp-Logan phantom is a popular mathematical model of a cranial slice, made up of a set of ellipses. This allows rigorous testing of computed tomography (CT) algorithms as it can be analytically transformed with the radon transform.

Parameters

- **N** (*int*) – The edge length of the square image to be produced
- **phantom_type** (*str, optional*) – The type of phantom to produce. Either `Modified Shepp-Logan` or `Shepp-Logan`. The default value is `Modified Shepp-Logan`. This is overriden if `ellipses` is also specified.
- **ellipses** (*array like*) – Custom set of ellipses to use.

Note: To use ellipses, these should be in the form `[[I, a, b, x0, y0, phi], [I, a, b, x0, y0, phi], ...]` where each row defines an ellipse and:

- `I` : Additive intensity of the ellipse.
- `a` : Length of the major axis.
- `b` : Length of the minor axis.
- `x0` : Horizontal offset of the centre of the ellipse.
- `y0` : Vertical offset of the centre of the ellipse.
- `phi` : Counterclockwise rotation of the ellipse in degrees, measured as the angle between the horizontal axis and the ellipse major axis.

The image bouding box in the algorithm is `[-1, -1], [1, 1]`, so the values of `a`, `b`, `x0` and `y0` should all be specified with respect to this box.

Returns **P** – A 2-dimensional array containing th Shepp-Logan phantom image.

[1] Shepp, L. A., Logan, B. F., "Reconstructing Interior Head Tissue from X-Ray Transmission", IEEE Transactions on Nuclear Science, Feb. 1974, p. 232

[2] Toft, P., "The Radon Transform - Theory and Implementation", Ph.D. thesis, Department of Mathematical Modelling, Technical University of Denmark, June 1996

Return type ndarray

Examples

```
>>> import matplotlib.pyplot as plt
>>> P = phantom()
>>> # P = phantom(256, 'Modified Shepp-Logan', None)
>>> plt.imshow(P)
```

References

CHAPTER

NINE

TOUPY.TOMO PACKAGE

9.1 Submodules

9.2 toupy.tomo.iradon module

toupy.tomo.iradon.**backprojector**(*sinogram*, *theta*, ***params*)
Wrapper to choose between Forward Radon transform using Silx and OpenCL or standard reconstruction.

Parameters

- **sinogram** (*ndarray*) – A 2-dimensional array containing the sinogram
- **theta** (*ndarray*) – A 1-dimensional array of thetas
- **params** (*dict*) – Dictionary containing the parameters to be used in the reconstruction. See *mod_iradonSilx()* and *mod_iradon()* for the list of parameters

Returns recons – A 2-dimensional array containing the reconstructed sliced by the choosen method

Return type ndarray

toupy.tomo.iradon.**compute_angle_weights**(*theta*)
Compute the corresponding weight for each angle according to the distance between its neighbors in case of non equally spaced angles

Parameters theta (*ndarray*) – Angles in degrees

Returns weights – The weights for each angle to be applied to the sinogram

Return type ndarray

Note: The weights are computed assuming a angular distribution between 0 and 180 degrees. Forked from odtbrain.util.compute_angle_weights_1d (https://github.com/RI-imaging/ODTbrain/)

toupy.tomo.iradon.**compute_filter**(*nbins*, *filter_type='ram-lak'*, *derivatives=False*, *freqcutoff=1*)
Compute the filter for the FBP tomographic reconstruction

Parameters

- **nbins** (*int*) – Size of the filter to be calculated
- **filter_type** (*str, optional*) – Name of the filter to be applied. The options are: *ram-lak*, *shepp-logan*, *cosine*, *hamming*, *hann*. The default is *ram-lak*.
- **derivatives** (*bool, optional*) – If True, it will use a Hilbert filter used for derivative projections. The default is `True``.
- **freqcutoff** (*float, optional*) – Normalized frequency cutoff of the filter. The default value is 1 which means no cutoff.

Returns **fourier_filter** – A 2-Dimnesional array containing the filter to be used in the FBP reconstruction

Return type ndarray

`toupy.tomo.iradon.`**`mod_iradon`**(*radon_image*, *theta=None*, *output_size=None*, *filter_type='ram-lak'*, *derivatives=False*, *interpolation='linear'*, *circle=False*, *freqcutoff=1*)

Inverse radon transform.

Reconstruct an image from the radon transform, using the filtered back projection algorithm.

Parameters

- **radon_image** (*ndarray*) – A 2-dimensional array containing radon transform (sinogram). Each column of the image corresponds to a projection along a different angle. The tomography rotation axis should lie at the pixel index `radon_image.shape[0] // 2` along the 0th dimension of `radon_image`.
- **theta** (*ndarray, optional*) – Reconstruction angles (in degrees). Default: m angles evenly spaced between 0 and 180 (if the shape of *radon_image* is (N, M)).
- **output_size** (*int*) – Number of rows and columns in the reconstruction.
- **filter** (*str, optional*) – Name of the filter to be applied in frequency domain filtering. The options are: *ram-lak*, *shepp-logan*, *cosine*, *hamming*, *hann*. The default is *ram-lak*. Assign None to use no filter.
- **derivatives** (*bool, optional*) – If `True`, assumes that the radon_image contains the derivates of the projections. The default is `True`
- **interpolation** (*str, optional*) – Interpolation method used in reconstruction. Methods available: *linear*, *nearest*, and *cubic* (*cubic* is slow). The default is *linear*
- **circle** (*bool, optional*) – Assume the reconstructed image is zero outside the inscribed circle. Also changes the default output_size to match the behaviour of `radon` called with `circle=True`.
- **freqcutoff** (*int, optional*) – Normalized frequency cutoff of the filter. The default value is 1 which means no cutoff.

Returns **reconstructed** – A 2-dimensional array containing the reconstructed image. The rotation axis will be located in the pixel with indices `(reconstructed.shape[0] // 2, reconstructed.shape[1] // 2)`.

Return type ndarray

Notes

It applies the Fourier slice theorem to reconstruct an image by multiplying the frequency domain of the filter with the FFT of the projection data. This algorithm is called filtered back projection.

`toupy.tomo.iradon.`**`mod_iradonSilx`**(*radon_image*, *theta=None*, *output_size=None*, *filter_type='ram-lak'*, *derivatives=False*, *interpolation='linear'*, *circle=False*, *freqcutoff=1*, *use_numpy=True*)

Inverse radon transform using Silx and OpenCL.

Reconstruct an image from the radon transform, using the filtered back projection algorithm.

Parameters

- **radon_image** (*ndarray*) – A 2-dimensional array containing radon transform (sinogram). Each column of the image corresponds to a projection along a different angle. The tomography rotation axis should lie at the pixel index `radon_image.shape[0] // 2` along the 0th dimension of `radon_image`.
- **theta** (*ndarray, optional*) – Reconstruction angles (in degrees). Default: m angles evenly spaced between 0 and 180 (if the shape of *radon_image* is (N, M)).

- **output_size** (*int*) – Number of rows and columns in the reconstruction.
- **filter** (*str, optional*) – Name of the filter to be applied in frequency domain filtering. The options are: *ram-lak*, *shepp-logan*, *cosine*, *hamming*, *hann*. The default is *ram-lak*. Assign None to use no filter.
- **derivatives** (*bool, optional*) – If `True`, assumes that the radon_image contains the derivates of the projections. The default is `True`
- **interpolation** (*str, optional*) – Interpolation method used in reconstruction. Methods available: *linear*, *nearest*, and *cubic* (*cubic* is slow). The default is *linear*
- **circle** (*boolean, optional*) – Assume the reconstructed image is zero outside the inscribed circle. Also changes the default output_size to match the behaviour of `radon` called with `circle=True`.
- **freqcutoff** (*int, optional*) – Normalized frequency cutoff of the filter. The default value is 1 which means no cutoff.

Returns **reconstructed** – A 2-dimensional array containing the reconstructed image. The rotation axis will be located in the pixel with indices `(reconstructed.shape[0] // 2, reconstructed.shape[1] // 2)`.

Return type ndarray

Notes

It applies the Fourier slice theorem to reconstruct an image by multiplying the frequency domain of the filter with the FFT of the projection data. This algorithm is called filtered back projection.

`toupy.tomo.iradon.`**`reconsSART`**(*sinogram*, *theta*, *num_iter=2*, *FBPinitial_guess=True*, *relaxation_params=0.15*, ***params*)

Reconstruction with SART algorithm

Parameters

- **sinogram** (*ndarray*) – A 2-dimensional array containing the sinogram
- **theta** (*ndarray*) – A 1-dimensional array of thetas
- **num_iter** (*int, optional*) – Number of iterations of the SART algorithm. The default is `2`.
- **FBPinitial_guess** (*bool, optional*) – If the results of FBP reconstruction should be used as initial guess. The default value is `True`
- **relaxation_params** (*float, optional*) – Relaxation parameter of SART. The default value is `0.15`.

Returns **recons** – A 2-dimensional array containing the reconstructed sliced by SART

Return type ndarray

9.3 toupy.tomo.radon module

`toupy.tomo.radon.`**`projector`**(*recons*, *theta*, ***params*)

Wrapper to choose between Forward Radon transform using Silx and OpenCL or standard reconstruction.

Parameters

- **recons** (*ndarray*) – A 2-dimensional array containing the tomographic slice
- **theta** (*ndarry*) – A 1-dimensional array of thetas
- **params** (*dict*) – Dictionary of parameters to be used

- **params["opencl"]** (*bool*) – If True, it will perform the tomographic reconstruction using the opencl implementation of Silx.

Returns **sinogramcomp** – A 2-dimensional array containing the reprojected sinogram

Return type ndarray

`toupy.tomo.radon.radonSilx`(*recons*, *theta*)
Forward Radon transform using Silx and OpenCL

Parameters

- **recons** (*ndarray*) – A 2-dimensional array containing the tomographic slice
- **theta** (*ndarry*) – A 1-dimensional array of thetas

Returns **sinogramcomp** – A 2-dimensional array containing the reprojected sinogram

Return type ndarray

9.4 toupy.tomo.tomorecons module

`toupy.tomo.tomorecons.full_tomo_recons`(*input_stack*, *theta*, ***params*)
Full tomographic reconstruction

Parameters

- **input_stack** (*ndarray*) – A 3-dimensional array containing the stack of projections. The order should be `[projection_num, row, column]`
- **theta** (*ndarray*) – A 1-dimensional array of thetas
- **params** (*dict*) – Dictionary containing additional parameters
- **params["algorithm"]** (*str*) – Choice of algorithm. Two algorithm implemented: "FBP" and "SART"
- **params["slicenum"]** (*int*) – Slice number
- **params["filtertype"]** (*str*) – Filter to use for FBP
- **params["freqcutoff"]** (*float*) – Frequency cutoff (between 0 and 1)
- **params["circle"]** (*bool*) – Multiply the reconstructed slice by a circle to remove borders
- **params["derivatives"]** (*bool*) – If the projections are derivatives. Only for FBP.
- **params["calc_derivatives"]** (*bool*) – Calculate derivatives of the sinogram if not done yet.
- **params["opencl"]** (*bool*) – Implement the tomographic reconstruction in opencl as implemented in Silx
- **params["autosave"]** (*bool*) – Save the data at the end without asking
- **params["vmin_plot"]** (*float*) – Minimum value for the gray level at each display
- **params["vmax_plot"]** (*float*) – Maximum value for the gray level at each display
- **params["colormap"]** (*str*) – Colormap
- **params["showrecons"]** (*bool*) – If to show the reconstructed slices

Returns **Tomogram** – A 3-dimensional array containing the full reconstructed tomogram

Return type ndarray

`toupy.tomo.tomorecons.tomo_recons`(*sinogram*, *theta*, ***params*)

Wrapper to select tomographic algorithm

sinogram [ndarray] A 2-dimensional array containing the sinogram

theta [ndarray] A 1-dimensional array of thetas

params [dict] Dictionary containing additional parameters

params["algorithm"] [str] Choice of algorithm. Two algorithm implemented: "FBP" and "SART"

params["slicenum"] [int] Slice number

params["filtertype"] [str] Name of the filter to be applied in frequency domain filtering. The options are: *ram-lak*, *shepp-logan*, *cosine*, *hamming*, *hann*. Assign None to use no filter.

params["freqcutoff"] [float] Frequency cutoff (between 0 and 1)

params["circle"] [bool] Multiply the reconstructed slice by a circle to remove borders

params["weight_angles"] [bool] If *True*, weights each projection with a factor proportional to the angular distance between the neighboring projections.

$$\Delta\phi_0 \longmapsto \Delta\phi_j =$$

rac{phi_{j+1} - phi_{j-1}}{2}

params["derivatives"] [bool] If the projections are derivatives. Only for FBP.

params["calc_derivatives"] [bool] Calculate derivatives of the sinogram if not done yet.

params["opencl"] [bool] Implement the tomographic reconstruction in opencl as implemented in Silx

params["autosave"] [bool] Save the data at the end without asking

params["vmin_plot"] [float] Minimum value for the gray level at each display

params["vmax_plot"] [float] Maximum value for the gray level at each display

params["colormap"] [str] Colormap

params["showrecons"] [bool] If to show the reconstructed slices

recons [ndarray] A 2-dimensional array containing the reconstructed slice

CHAPTER

TEN

TOUPY.UTILS PACKAGE

10.1 Submodules

10.2 toupy.utils.FFT_utils module

toupy.utils.FFT_utils.**fastfftn**(*input_array*, ***kwargs*)
: Auxiliary function to use pyFFTW. It does the align, planning and apply FFTW transform

  Parameters **input_array** (*array_like*) – Array to be FFTWed

  Returns **fftw_array** – Fourier transformed array

  Return type array_like

toupy.utils.FFT_utils.**fastifftn**(*input_array*, ***kwargs*)
: Auxiliary function to use pyFFTW. It does the align, planning and apply inverse FFTW transform

  Parameters **input_array** (*array_like*) – Array to be FFTWed

  Returns **ifftw_array** – Inverse Fourier transformed array

  Return type array_like

toupy.utils.FFT_utils.**is_power2**(*num*)
: States if a number num is a power of two

toupy.utils.FFT_utils.**nextpow2**(*number*)
: Find the next power 2 of number for FFT

toupy.utils.FFT_utils.**nextpoweroftwo**(*number*)
: Returns next power of two following number

toupy.utils.FFT_utils.**padfft**(*input_array*, *pad_mode='reflect'*)
: Auxiliary function to pad arrays for Fourier transforms. It accepts 1D and 2D arrays.

  Parameters

  - **input_array** (*array_like*) – Array to be padded
  - **mode** (*str*) – Padding mode to treat the array borders. See numpy.pad for modes. The default value is *reflect*.

  Returns

  - **array_pad** (*array_like*) – Padded array
  - **N_pad** (*array_like*) – padded frequency coordinates
  - **padw** (*int, list of ints*) – pad width

toupy.utils.FFT_utils.**padrightside**(*nbins*)
: Returns pad_width for padding at the right side given a value of nbins The pad_width is calculated with next_fast_len function from *PyFFTW* package

toupy.utils.FFT_utils.**padwidthbothsides**(*nbins*)
Returns pad_width for padding both sides given a value of `nbins`

10.3 toupy.utils.array_utils module

toupy.utils.array_utils.**create_circle**(*inputimg*)
Create circle with apodized edges

Parameters **inputimg** (*array_like*) – Input image from which to calculate the circle

Returns **t** – Array containing the circle

Return type array_like

toupy.utils.array_utils.**create_mask_borders**(*tomogram*, *mask_array*, *threshold=4e-07*)
Create mask for border of tomographic volume

Parameters

- **tomogram** (*array_like*) – Input volume
- **mask** (*bool array_like*) – Input mask
- **threshold** (*float, optional*) – Threshold value. The default value is `4e-7`.

Returns **mask_array** – Masked array

Return type array_like

toupy.utils.array_utils.**crop**(*input_array*, *delcropx*, *delcropy*)
Crop images

Parameters

- **input_array** (*array_like*) – Input image to be cropped
- **delcropx** (*int*) – amount of pixel to be cropped in x
- **delcropy** (*int*) – amount of pixel to be cropped in y

Returns Cropped image

Return type array_like

toupy.utils.array_utils.**cropROI**(*input_array*, *roi=[]*)
Crop ROI

Parameters

- **input_array** (*array_like*) – Input image to be cropped
- **roi** (*list of int*) – ROI of interest. roi should be [top, bottom, left, right]

Returns Cropped image

Return type array_like

toupy.utils.array_utils.**fract_hanning**(*outputdim*, *unmodsize*)
Creates a square hanning window if unmodsize = 0 (or ommited), otherwise the output array will contain an array of ones in the center and cosine modulation on the edges, the array of ones will have DC in upper left corner.

Parameters

- **outputdim** (*int*) – Size of the output array
- **unmodsize** (*int*) – Size of the central array containing no modulation.

Returns Square array containing a fractional separable Hanning window with DC in upper left corner.

Return type array_like

`toupy.utils.array_utils.`**`fract_hanning_pad`**(*outputdim*, *filterdim*, *unmodsize*)

Creates a square hanning window if unmodsize = 0 (or ommited), otherwise the output array will contain an array of ones in the center and cosine modulation on the edges, the array of ones will have DC in upper left corner.

Parameters

- **outputdim** (*int*) – Size of the output array
- **filterdim** (*int*) – Size of filter (it will zero pad if filterdim < outputdim)
- **unmodsize** (*int*) – Size of the central array containing no modulation.

Returns Square array containing a fractional separable Hanning window with DC in upper left corner.

Return type array_like

`toupy.utils.array_utils.`**`gauss_kern`**(*size*, *sizey=None*)

Returns a normalized 2D gauss kernel array for convolutions

Parameters

- **size** (*int*) – Size of the kernel
- **sizey** (*int, optional*) – Vertical size of the kernel if not squared

Returns Normalized kernel

Return type array_like

Notes

from: http://scipy.org/Cookbook/SignalSmooth

`toupy.utils.array_utils.`**`hanning_apod1D`**(*window_size*, *apod_width*)

Create 1D apodization window using Hanning window

Parameters

- **window_size** (*int*) – Window size
- **apod_width** (*int*) – Apodization width

Returns **hannwindow1D** – 1D Hanning window for the apodization

Return type array_like

`toupy.utils.array_utils.`**`hanning_apodization`**(*window_size*, *apod_width*)

Create apodization window using Hanning window

Parameters

- **window_size** (*tuple*) – Window size
- **apod_width** (*int*) – Apodization width

Returns **hannwindow2D** – 2D Hanning window for the apodization

Return type array_like

`toupy.utils.array_utils.`**`mask_borders`**(*imgarray*, *mask_array*, *threshold=4e-07*)

Mask borders using the gradient

Parameters

- **imgarray** (*array_like*) – Input image

- **mask_array** (*bool array_like*) – Input mask
- **threshold** (*float, optional*) – Threshold value. The default value is `4e-7`.

Returns **mask_array** – Masked array

Return type array_like

`toupy.utils.array_utils.`**`normalize_array`**(*input_array*)
Normalize the input array

`toupy.utils.array_utils.`**`padarray_bothsides`**(*input_array*, *newshape*, *padmode='edge'*)
Pad array in both sides

Parameters

- **input_array** (*array_like*) – Input array
- **newshape** (*tuple*) – New shape of the array to be padded
- **padmode** (*str*) – Padding mode. The default is `edge`

Returns Padded array

Return type array_like

`toupy.utils.array_utils.`**`polynomial1d`**(*x*, *order=1*, *w=1*)
Generates a 1D orthonormal polynomial base.

Parameters

- **x** (*array_like*) – Array containing the values of `x` for the polynomial
- **order** (*int, optional*) – Order of the polynomial. The defaul value is 1.
- **w** (*int, optional*) – Weights of the coefficients. The defaul value is 1.

Returns **polyseries** – Orthonormal polymonial up to order

Return type array_like

Note: Inspired by legendrepoly1D_2.m created by Manuel Guizar in March 10,2009

`toupy.utils.array_utils.`**`projectpoly1d`**(*func1d*, *order=1*, *w=1*)
Projects a 1D function onto orthonormalized base

Parameters

- **func1d** (*array_like*) – Array containing the values of the 1D function
- **order** (*int, optional*) – Order of the polynomial. The defaul value is 1.
- **w** (*int, optional*) – Weights of the coefficients. The defaul value is 1.

Returns **projfunc1d** – Projected 1D funtion on orthonormal base

Return type array_like

Note: Inspired by projectleg1D_2.m created by Manuel Guizar in March 10,2009

`toupy.utils.array_utils.`**`radtap`**(*X*, *Y*, *tappix*, *zerorad*)
Creates a central cosine tapering for beam. It receives the X and Y coordinates, tappix is the extent of tapering, zerorad is the radius with no data (zeros).

`toupy.utils.array_utils.`**`replace_bad`**(*input_stack*, *list_bad=[]*, *temporary=False*)
correcting bad projections before unwrapping

Parameters

- **input_stack** (*array_like*) – Stack of projections

- **list_bad** (*list*) – List of bad projections
- **temporary** (*bool*) – If *False*, the projection will be interpolated with the previous and after projections. If *True*, the projection will be replaced by the previous projection.

`toupy.utils.array_utils.`**`round_to_even`**(*x*)
Round number `x` to next even number

`toupy.utils.array_utils.`**`sharpening_image`**(*input_image*, *filter_size=3*, *alpha=30*)
Sharpen image with a median filter

Parameters

- **input_image** (*array_like*) – Image to be sharpened
- **filter_size** (*int*) – Size of the filter
- **alpha** (*float*) – Strength of the sharpening

Returns Sharpened image

Return type array_like

`toupy.utils.array_utils.`**`smooth1d`**(*x*, *window_len=11*, *window='hanning'*)
Smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

Parameters

- **x** (*array_like*) – The input signal
- **window_len** (*int, optional,*) – The dimension of the smoothing window; should be an odd integer. The default value is *11*.
- **window** (*str, optional*) – The type of window from *flat*, *hanning*, *hamming*, *bartlett*, *blackman* flat window will produce a moving average smoothing.

Returns **y** – The smoothed signal

Return type array_like

Example

```
>>> import numpy as np
>>> t=np.linspace(-2,2,0.1)
>>> x=np.sin(t)+np.random.randn(len(t))*0.1
>>> y=smooth(x)
```

Notes

see also: numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve

scipy.signal.lfilter

Adapted from : https://scipy-cookbook.readthedocs.io/items/SignalSmooth.html from: http://scipy.org/Cookbook/SignalSmooth

`toupy.utils.array_utils.`**`smooth2d`**(*im*, *n*, *ny=None*)

Blurs the image by convolving with a gaussian kernel of typical

size n. The optional keyword argument ny allows for a different size in the y direction.

Parameters

- **im** (*array_like*) – Input image
- **n** (*int, optional*) – Typical size of the gaussian kernel
- **n** – Size in the y direction if not squared

Returns **improc** – Smoothed image

Return type array_like

Notes

from: http://scipy.org/Cookbook/SignalSmooth

toupy.utils.array_utils.**smooth_image**(*input_image*, *filter_size=3*)
Smooth image with a median filter

Parameters

- **input_image** (*array_like*) – Image to be smoothed
- **filter_size** (*int*) – Size of the filter

Returns Smoothed image

Return type array_like

toupy.utils.array_utils.**sort_array**(*input_array*, *ref_array*)
Sort array based on another array

Parameters

- **input_array** (*array_like*) – Array to be sorted
- **ref_array** (*array_like*) – Array on which the sorting will be based

Returns

- **sorted_input_array** (*array_like*) – Sorted input array
- **sorted_ref_array** (*array_like*) – Sorted reference array

10.4 toupy.utils.converter_utils module

toupy.utils.converter_utils.**convert_to_beta**(*input_img*, *energy*, *voxelsize*, *apply_log=False*)
Converts the image gray-levels from amplitude to beta

toupy.utils.converter_utils.**convert_to_delta**(*input_img*, *energy*, *voxelsize*)
Converts the image gray-levels from phase-shifts to delta

toupy.utils.converter_utils.**convert_to_mu**(*input_img*, *wavelen*)
Converts the image gray-levels from absoption index Beta to linear attenuation coefficient mu

toupy.utils.converter_utils.**convert_to_rhoe**(*input_img*, *wavelen*)
Converts the image gray-levels from delta to electron density

toupy.utils.converter_utils.**convert_to_rhom**(*input_img*, *wavelen*, *A*, *Z*)
Converts the image gray-levels from electron density to mass density

10.5 toupy.utils.fit_utils module

toupy.utils.fit_utils.**model_erf**(*t*, **coeffs*)

Model for the erf fitting

P0 + P1*t + (P2/2)*(1-erf(sqrt(2)*(x-P3)/(P4)))

Parameters

- **t** (*ndarray*) – Input coordinates
- **coeffs[0]** (*float*) – P0 (noise)
- **coeffs[1]** (*float*) – P1 (linear term)
- **coeffs[2]** (*float*) – P2 (Maximum amplitude)
- **coeffs[3]** (*float*) – P3 (center)
- **coeffs[4]** (*float*) – P4 (width)

Returns Array containing the model

Return type ndarray

toupy.utils.fit_utils.**model_tanh**(*t*, **coeffs*)

Model for the erf fitting

P0 + P1*t + (P2/2)*(1-tanh(sqrt(2)*(x-P3)/P4))

Parameters

- **t** (*ndarray*) – Input coordinates
- **coeffs[0]** (*float*) – P0 (noise)
- **coeffs[1]** (*float*) – P1 (linear term)
- **coeffs[2]** (*float*) – P2 (Maximum amplitude)
- **coeffs[3]** (*float*) – P3 (center)
- **coeffs[4]** (*float*) – P4 (width)

Returns Array containing the model

Return type ndarray

toupy.utils.fit_utils.**residuals_erf**(*coeffs*, *y*, *t*)

Residuals for the least-squares optimization coeffs as the ones of the model erf function

Parameters

- **y** (*ndarray*) – The data
- **t** (*ndarray*) – Input coordinates

Returns Residuals

Return type ndarray

toupy.utils.fit_utils.**residuals_tanh**(*coeffs*, *y*, *t*)

Residuals for the least-squares optimization coeffs as the ones of the model tanh function

Parameters

- **y** (*ndarray*) – The data
- **t** (*ndarray*) – Input coordinates

Returns Residuals

Return type ndarray

10.6 toupy.utils.funcutils module

`toupy.utils.funcutils.`**`checkhostname`**`(`*func*`)`
Check if running in OAR, if not, exit.

`toupy.utils.funcutils.`**`deprecated`**`(`*func*`)`
This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

`toupy.utils.funcutils.`**`downloadURL`**`(`*url*, *fname*`)`
Download file from a URL.

Parameters

- **url** (*str*) – URL address
- **fname** (*str*) – Filename as to be stored

`toupy.utils.funcutils.`**`downloadURLfile`**`(`*url*, *filename*`)`
Download and save file from a URL.

Parameters

- **url** (*str*) – URL address
- **fname** (*str*) – Filename as to be stored

`toupy.utils.funcutils.`**`progbar`**`(`*curr*, *total*, *textstr=''*`)`
Create a progress bar for for-loops.

Parameters

- **curr** (*int*) – Current value to shown in the progress bar
- **total** (*int*) – Maximum size of the progress bar.
- **textstr** (*str*) – String to be shown at the right side of the progress bar

class `toupy.utils.funcutils.`**`switch`**`(`*value*`)`
Bases: `object`

This class provides the functionality of switch or case in other languages than python. This mimics the functionality of *switch* in Python

`__iter__``()`
Return the match method once, then stop

`match``(`**args*`)`
Indicate whether or not to enter a case suite

10.7 toupy.utils.plot_utils module

class `toupy.utils.plot_utils.`**`RegisterPlot`**`(`***params*`)`
Bases: `object`

Display plots during registration

`plotshorizontal``(`*recons*, *sinoorig*, *sinocurr*, *sinocomp*, *deltaslice*, *metric_error*, *count*`)`
Display plots during the horizontal registration

`plotsvertical``(`*proj*, *lims*, *vertfluctinit*, *vertfluctcurr*, *deltastack*, *metric_error*, *count*`)`
Display plots during the vertical registration

`updatehorizontal``()`
Update the plot canvas during horizontal registration

updatevertical()
Update the plot canvas during vertical registration

class toupy.utils.plot_utils.**ShowProjections**
Bases: `object`

Show projections and probe

static probe2HSV(*probe*)
Special tricks for the probe display in HSV

show_projections(*obj*, *probe*, *idxp*)
Show the object and the probe :param obj: Object to show :type obj: ndarray :param probe: Probe to show :type probe: ndarray :param idxp: Projection number :type idxp: int

update_show()
Update the canvas

toupy.utils.plot_utils.**animated_image**(*stack_array*, **args*)
Iterative plot of the images using animation module of Matplotlib

Parameters

- **stack_array** (*ndarray*) – Array containing the stack of images to animate. The first index corresponds to the image number in the sequence of images.
- **args[0]** (*list of ints*) – Row limits to display
- **args[1]** (*list of ints*) – Column limits to display

toupy.utils.plot_utils.**autoscale_y**(*ax*, *margin=0.1*)
This function rescales the y-axis based on the data that is visible given the current xlim of the axis.

Parameters

- **ax** (*object*) – A matplotlib axes object
- **margin** (*float*) – The fraction of the total height of the y-data to pad the upper and lower ylims

toupy.utils.plot_utils.**display_slice**(*recons*, *colormap='bone'*, *vmin=None*, *vmax=None*)
Display tomographic slice

Parameters

- **recons** (*array_like*) – Tomographic slice
- **colormap** (*str, optional*) – Colormap name. The default value is bone
- **vmin** (*float, None*) – Minimum gray-level. The default value is None
- **vmax** (*float, None*) – Maximum gray-level. The default value is None

toupy.utils.plot_utils.**isnotebook**()
Check if code is executed in the IPython notebook. This is important because jupyter notebook does not support iterative plots

toupy.utils.plot_utils.**iterative_show**(*stack_array*, *limrow=[]*, *limcol=[]*, *airpixel=[]*, *onlyroi=False*, *colormap='bone'*, *vmin=None*, *vmax=None*)
Iterative plot of the images

Parameters

- **stack_array** (*ndarray*) – Array containing the stack of images to animate. The first index corresponds to the image number in the sequence of images.
- **limrow** (*list of ints*) – Limits of rows in the format [begining, end]
- **limcol** (*list of ints*) – Limits of cols in the format [begining, end]
- **airpixel** (*list of ints*) – Position of pixel in the air/vacuum

- **onlyroi** (*bool*) – If True, it displays only the ROI. If False, it displays the entire image.
- **colormap** (*str, optional*) – Colormap name. The default value is bone
- **vmin** (*float, None, optional*) – Minimum gray-level. The default value is None
- **vmax** (*float, None, optional*) – Maximum gray-level. The default value is None

toupy.utils.plot_utils.**plot_checkangles**(*angles*)
Plot the angles for each projections and the derivatives to check for anomalies

Parameters **angles** (*array_like*) – Array of angles

toupy.utils.plot_utils.**show_linearphase**(*image*, *mask*, **args*)
Show projections and probe

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

M

N

O

P

R

S

T

U

V

W