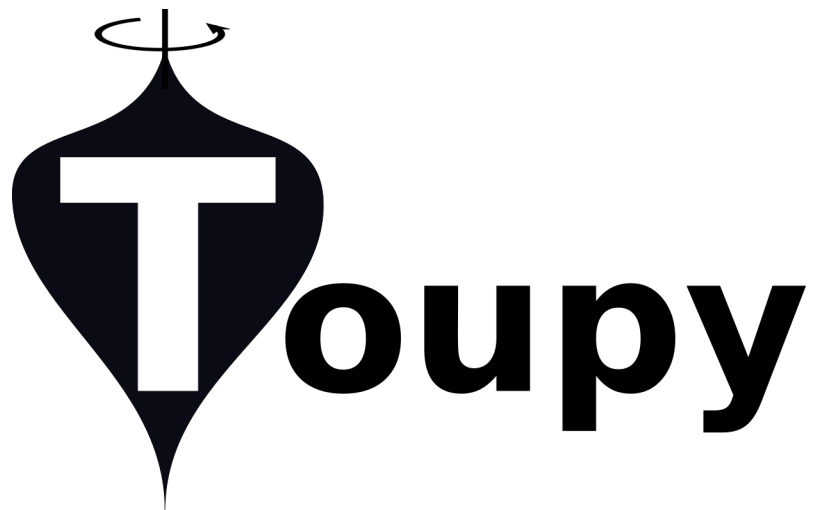




# **Toupy Documentation**

*Release 0.1.0*

**Julio C. da Silva**

**May 31, 2022**



# INTRODUCTION

|          |                                                    |           |
|----------|----------------------------------------------------|-----------|
| <b>1</b> | <b>Welcome to toupy documentation</b>              | <b>1</b>  |
| 1.1      | Quicklinks . . . . .                               | 1         |
| 1.2      | References . . . . .                               | 1         |
| 1.3      | Acknowledgments . . . . .                          | 1         |
| <b>2</b> | <b>Installation</b>                                | <b>3</b>  |
| 2.1      | Requirements . . . . .                             | 3         |
| 2.2      | Via pip install . . . . .                          | 3         |
| 2.3      | Via Python Package . . . . .                       | 3         |
| <b>3</b> | <b>Templates</b>                                   | <b>5</b>  |
| 3.1      | Tomographic Reconstruction . . . . .               | 5         |
| <b>4</b> | <b>toupy.io package</b>                            | <b>9</b>  |
| 4.1      | Submodules . . . . .                               | 9         |
| 4.2      | toupy.io.dataio module . . . . .                   | 9         |
| 4.3      | toupy.io.filesrw module . . . . .                  | 15        |
| 4.4      | toupy.io.h5chunk_shape_3D module . . . . .         | 20        |
| <b>5</b> | <b>toupy.registration package</b>                  | <b>23</b> |
| 5.1      | Submodules . . . . .                               | 23        |
| 5.2      | toupy.registration.registration module . . . . .   | 23        |
| 5.3      | toupy.registration.shift module . . . . .          | 27        |
| <b>6</b> | <b>toupy.resolution package</b>                    | <b>29</b> |
| 6.1      | Submodules . . . . .                               | 29        |
| 6.2      | toupy.resolution.FSC module . . . . .              | 29        |
| 6.3      | toupy.resolution.FSCtools module . . . . .         | 30        |
| <b>7</b> | <b>toupy.restoration package</b>                   | <b>33</b> |
| 7.1      | Submodules . . . . .                               | 33        |
| 7.2      | toupy.restoration.GUI_tracker module . . . . .     | 33        |
| 7.3      | toupy.restoration.derivativetools module . . . . . | 35        |
| 7.4      | toupy.restoration.ramptools module . . . . .       | 37        |
| 7.5      | toupy.restoration.roipoly module . . . . .         | 38        |
| 7.6      | toupy.restoration.unwraptools module . . . . .     | 38        |
| 7.7      | toupy.restoration.vortices tools module . . . . .  | 41        |
| <b>8</b> | <b>toupy.simulation package</b>                    | <b>43</b> |
| 8.1      | Submodules . . . . .                               | 43        |
| 8.2      | toupy.simulation.phantom_creator module . . . . .  | 43        |
| <b>9</b> | <b>toupy.tomo package</b>                          | <b>45</b> |
| 9.1      | Submodules . . . . .                               | 45        |
| 9.2      | toupy.tomo.iradon module . . . . .                 | 45        |

|           |                                              |           |
|-----------|----------------------------------------------|-----------|
| 9.3       | toupy.tomo.radon module . . . . .            | 48        |
| 9.4       | toupy.tomo.tomorecons module . . . . .       | 48        |
| <b>10</b> | <b>toupy.utils package</b>                   | <b>51</b> |
| 10.1      | Submodules . . . . .                         | 51        |
| 10.2      | toupy.utils.FFT_utils module . . . . .       | 51        |
| 10.3      | toupy.utils.array_utils module . . . . .     | 52        |
| 10.4      | toupy.utils.converter_utils module . . . . . | 57        |
| 10.5      | toupy.utils.fit_utils module . . . . .       | 57        |
| 10.6      | toupy.utils.funcutils module . . . . .       | 59        |
| 10.7      | toupy.utils.plot_utils module . . . . .      | 59        |
| <b>11</b> | <b>Indices and tables</b>                    | <b>63</b> |
|           | <b>Python Module Index</b>                   | <b>65</b> |
|           | <b>Index</b>                                 | <b>67</b> |

## 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<sup>1</sup>, but which is also based on previous works<sup>2</sup>.

### 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

---

<sup>1</sup> 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](#)

<sup>2</sup> 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](#)



## 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
```



## 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 depends 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 opens 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.



## TOUPY.IO PACKAGE

### 4.1 Submodules

### 4.2 `toupy.io.dataio` module

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

Bases: `PathName`, `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 additional 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 additional 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: *PathName*, 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: [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: *PathName*, *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: *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 corresponding 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, detector='Frelon')`

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='tiffle')`

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<sup>1</sup>

**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)

---

<sup>1</sup> [https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking\\_data\\_choosing\\_shapes](https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking_data_choosing_shapes)

**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<sup>2</sup> and<sup>3</sup>.

**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  $2^{*\text{len}(\text{shape})}$

---

<sup>2</sup> [https://www.unidata.ucar.edu/blog\\_content/data/2013/chunk\\_shape\\_3D.py](https://www.unidata.ucar.edu/blog_content/data/2013/chunk_shape_3D.py)

<sup>3</sup> <https://github.com/HDFGroup/datacontainer/blob/master/lib/chunking.py>



## 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<sup>1,2</sup>. 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 containing the aligned sinogram

---

<sup>1</sup> Guizar-Sicairos, M., et al., "Quantitative interior x-ray nanotomography by a hybrid imaging technique," *Optica* 2, 259-266 (2015).

<sup>2</sup> 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<sup>3,4</sup>. 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 object 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

---

<sup>3</sup> Guizar-Sicairos, M., et al. , “Phase tomography from x-ray coherent diffractive imaging projections,” Opt. Express 19, 21345-21357 (2011).

<sup>4</sup> 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)

**Return type**

array\_like

```
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 alignment 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: 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 wrapping) 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 wrapping) 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

## 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: `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<sup>1</sup> between *image1* and *image2*, and estimate the resolution based on the threshold function *T* of 1 or 1/2 bit.

#### Parameters

---

<sup>1</sup> 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  $SNR_t = 0.5$ , which should be used for two independent measurements. The option *halfbit* means 1/2 bit threshold with  $SNR_t = 0.2071$ , 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



## 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: [`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, refer 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 additional 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 additional 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 *None* 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 Ptycho.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<sup>1</sup> for a given wrapped phase image.

**Parameters**

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

**Returns**

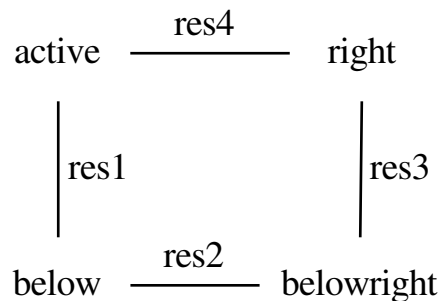
**residues** – A 2-dimensional array containing the map of residues (valued +1 or -1)

**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

---

## 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**

---

<sup>1</sup> R. M. Goldstein, H. A. Zebker and C. L. Werner, Radio Science 23, 713-720 (1988).

- **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.unwrap_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["vmax"]** – 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 Herraiez et al.<sup>2</sup> implemented in Scikit-Image (<https://scikit-image.org>).

---

**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>

---

---

<sup>2</sup> Miguel Arevallilo Herraiez, 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

`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.vortices tools module

`toupy.restoration.vortices tools.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.vortices tools.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.vortices.tools.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.vortices.tools.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.vortices.tools.rm_vortices_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.vortices.tools.rm_vortices_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.

---

## 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<sup>1</sup> or modified Shepp-Logan phantom<sup>2</sup>. 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 overridden 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 bounding 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.

---

<sup>1</sup> Shepp, L. A., Logan, B. F., “Reconstructing Interior Head Tissue from X-Ray Transmission”, IEEE Transactions on Nuclear Science, Feb. 1974, p. 232

<sup>2</sup> Toft, P., “The Radon Transform - Theory and Implementation”, Ph.D. thesis, Department of Mathematical Modelling, Technical University of Denmark, June 1996

### Returns

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

### Return type

ndarray

### Examples

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

### References

## 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`.
- **frecutoff** (*float*, *optional*) – Normalized frequency cutoff of the filter. The default value is 1 which means no cutoff.

**Returns**

**fourier\_filter** – A 2-Dimensional 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, frecutoff=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 derivatives 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`.
- **frecutoff** (*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 derivatives 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** (*ndarray*) – 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** (*ndarray*) – 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.

$$\Delta\phi_0 \mapsto \Delta\phi_j =$$

$\text{rac}\{\text{phi}_{\{j+1\}} - \text{phi}_{\{j-1\}}\}\{2\}$

**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

## 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 omitted), 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 omitted), 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_bothersides(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 default value is 1.
- **w** (*int*, *optional*) – Weights of the coefficients. The default value is 1.

#### Returns

**polyseries** – Orthonormal polynomial 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 default value is 1.
- **w** (*int*, *optional*) – Weights of the coefficients. The default value is 1.

#### Returns

**projfunc1d** – Projected 1D function 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 beginning 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
- **ny** – 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 absorption 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

$P0 + P1 * t + (P2/2) * (1 - \text{erf}(\sqrt{2} * (x - P3)/(P4)))$

### 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



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### t

- `toupy.io`, 9
- `toupy.io.dataio`, 9
- `toupy.io.filesrw`, 15
- `toupy.io.h5chunk_shape_3D`, 20
- `toupy.registration`, 23
- `toupy.registration.registration`, 23
- `toupy.registration.shift`, 27
- `toupy.resolution`, 29
- `toupy.resolution.FSC`, 29
- `toupy.resolution.FSCtools`, 30
- `toupy.restoration`, 33
- `toupy.restoration.derivativetools`, 35
- `toupy.restoration.GUI_tracker`, 33
- `toupy.restoration.ramptools`, 37
- `toupy.restoration.roipoly`, 38
- `toupy.restoration.unwraptools`, 38
- `toupy.restoration.vortices tools`, 41
- `toupy.simulation`, 43
- `toupy.simulation.phantom_creator`, 43
- `toupy.tomo`, 45
- `toupy.tomo.iradon`, 45
- `toupy.tomo.radon`, 48
- `toupy.tomo.tomorecons`, 48
- `toupy.utils`, 51
- `toupy.utils.array_utils`, 52
- `toupy.utils.converter_utils`, 57
- `toupy.utils.FFT_utils`, 51
- `toupy.utils.fit_utils`, 57
- `toupy.utils.funcutils`, 59
- `toupy.utils.plot_utils`, 59



## Symbols

`__all__` (in module `toupy.io.h5chunk_shape_3D`), 20  
`__call__` () (in module `toupy.registration.shift.ShiftFunc` method), 27  
`__iter__` () (in module `toupy.utils.funcutils.switch` method), 59

## A

`add_mask` () (in module `toupy.restoration.GUI_tracker.PhaseTracker` method), 33  
`alignprojections_horizontal` () (in module `toupy.registration.registration`), 23  
`alignprojections_vertical` () (in module `toupy.registration.registration`), 24  
`AmpTracker` (class in `toupy.restoration.GUI_tracker`), 33  
`animated_image` () (in module `toupy.utils.plot_utils`), 60  
`apodization` () (in module `toupy.resolution.FSC.FourierShellCorr` method), 30  
`apply_all_masks` () (in module `toupy.restoration.GUI_tracker.AmpTracker` method), 33  
`apply_all_masks` () (in module `toupy.restoration.GUI_tracker.PhaseTracker` method), 33  
`apply_mask` () (in module `toupy.restoration.GUI_tracker.AmpTracker` method), 33  
`apply_mask` () (in module `toupy.restoration.GUI_tracker.PhaseTracker` method), 33  
`autoscale_y` () (in module `toupy.utils.plot_utils`), 60

## B

`backprojector` () (in module `toupy.tomo.iradon`), 45  
`binlist` () (in module `toupy.io.h5chunk_shape_3D`), 20

## C

`calculate_derivatives` () (in module `toupy.restoration.derivativetools`), 35  
`calculate_derivatives_fft` () (in module `toupy.restoration.derivativetools`), 35  
`cart2pol` () (in module `toupy.restoration.vortices_tools`), 41  
`center_of_mass_stack` () (in module `toupy.registration.registration`), 24  
`check_angles` () (in module `toupy.io.dataio.LoadProjections` method), 11

`check_angles_new` () (in module `toupy.io.dataio.LoadProjections` method), 11  
`checkhostname` () (in module `toupy.utils.funcutils`), 59  
`chooseregiontoderivatives` () (in module `toupy.restoration.derivativetools`), 36  
`chooseregiontounwrap` () (in module `toupy.restoration.unwrap_tools`), 38  
`chunk_shape_3D` () (in module `toupy.io.h5chunk_shape_3D`), 20  
`circle` () (in module `toupy.resolution.FSC.FourierShellCorr` method), 30  
`cmvmax` () (in module `toupy.restoration.GUI_tracker.PhaseTracker` method), 33  
`cmvmin` () (in module `toupy.restoration.GUI_tracker.PhaseTracker` method), 33  
`compute_2tomograms` () (in module `toupy.resolution.FSCtools`), 30  
`compute_2tomograms_split` () (in module `toupy.resolution.FSCtools`), 31  
`compute_aligned_horizontal` () (in module `toupy.registration.registration`), 24  
`compute_aligned_sino` () (in module `toupy.registration.registration`), 25  
`compute_aligned_stack` () (in module `toupy.registration.registration`), 25  
`compute_angle_weights` () (in module `toupy.tomo.iradon`), 45  
`compute_filter` () (in module `toupy.tomo.iradon`), 45  
`convert16bitstiff` () (in module `toupy.io.filesrw`), 15  
`convert8bitstiff` () (in module `toupy.io.filesrw`), 15  
`convert_to_beta` () (in module `toupy.utils.converter_utils`), 57  
`convert_to_delta` () (in module `toupy.utils.converter_utils`), 57  
`convert_to_mu` () (in module `toupy.utils.converter_utils`), 57  
`convert_to_rhoe` () (in module `toupy.utils.converter_utils`), 57  
`convert_to_rho` () (in module `toupy.utils.converter_utils`), 57  
`convert_to_tiff` () (in module `toupy.io.dataio.SaveTomogram` class method), 14  
`convertimageto16bits` () (in module `toupy.io.filesrw`), 15

`convertimageto8bits()` (in module `toupy.io.filesrw`), 15  
`create_circle()` (in module `toupy.utils.array_utils`), 52  
`create_mask_borders()` (in module `toupy.utils.array_utils`), 52  
`create_paramsh5()` (in module `toupy.io.filesrw`), 16  
`crop()` (in module `toupy.utils.array_utils`), 52  
`crop_array()` (in module `toupy.io.filesrw`), 16  
`cropROI()` (in module `toupy.utils.array_utils`), 52

## D

`datafilewcard()` (`toupy.io.dataio.PathName` method), 13  
`deprecated()` (in module `toupy.utils.funcutils`), 59  
`derivatives()` (in module `toupy.restoration.derivativetools`), 36  
`derivatives_fft()` (in module `toupy.restoration.derivativetools`), 36  
`derivatives_sino()` (in module `toupy.restoration.derivativetools`), 36  
`display_slice()` (in module `toupy.utils.plot_utils`), 60  
`displayMean()` (`toupy.restoration.roipoly.roipoly` method), 38  
`displayROI()` (`toupy.restoration.roipoly.roipoly` method), 38  
`distance()` (in module `toupy.restoration.unwraptools`), 38  
`down()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 33  
`downloadURL()` (in module `toupy.utils.funcutils`), 59  
`downloadURLfile()` (in module `toupy.utils.funcutils`), 59  
`draw_mask()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 33

## E

`estimate_rot_axis()` (in module `toupy.registration.registration`), 25

## F

`fastfftn()` (in module `toupy.utils.FFT_utils`), 51  
`fastifftn()` (in module `toupy.utils.FFT_utils`), 51  
`fouriercorr()` (`toupy.resolution.FSC.FourierShellCorr` method), 30  
`FourierShellCorr` (class in `toupy.resolution.FSC`), 29  
`fract_hanning()` (in module `toupy.utils.array_utils`), 53  
`fract_hanning_pad()` (in module `toupy.utils.array_utils`), 53  
`FSCPlot` (class in `toupy.resolution.FSC`), 29  
`full_tomo_recons()` (in module `toupy.tomo.tomorecons`), 48

## G

`gauss_kern()` (in module `toupy.utils.array_utils`), 53

`get_charge()` (in module `toupy.restoration.unwraptools`), 38  
`get_object_novort()` (in module `toupy.restoration.vorticestools`), 41  
`get_probe_novort()` (in module `toupy.restoration.vorticestools`), 41  
`getMask()` (`toupy.restoration.roipoly.roipoly` method), 38  
`gradient_axis()` (in module `toupy.restoration.derivativetools`), 36  
`gui_plotamp()` (in module `toupy.restoration.GUI_tracker`), 34  
`gui_plotphase()` (in module `toupy.restoration.GUI_tracker`), 35

## H

`hanning_apod1D()` (in module `toupy.utils.array_utils`), 54  
`hanning_apodization()` (in module `toupy.utils.array_utils`), 54

## I

`insert_missing()` (`toupy.io.dataio.LoadProjections` static method), 11  
`is_power2()` (in module `toupy.utils.FFT_utils`), 51  
`isnotebook()` (in module `toupy.utils.plot_utils`), 60  
`iterative_show()` (in module `toupy.utils.plot_utils`), 60

## K

`key_event()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 33

## L

`load()` (`toupy.io.dataio.LoadData` class method), 9  
`load()` (`toupy.io.dataio.LoadProjections` class method), 11  
`load()` (`toupy.io.dataio.LoadTomogram` class method), 12  
`load_masks()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`load_olddata()` (`toupy.io.dataio.LoadData` class method), 10  
`load_paramsh5()` (in module `toupy.io.filesrw`), 16  
`LoadData` (class in `toupy.io.dataio`), 9  
`loadedf()` (`toupy.io.dataio.LoadProjections` class method), 12  
`loadmasks()` (`toupy.io.dataio.LoadData` class method), 10  
`LoadProjections` (class in `toupy.io.dataio`), 11  
`loadshiftstack()` (`toupy.io.dataio.LoadData` class method), 11  
`loadtheta()` (`toupy.io.dataio.LoadData` class method), 11  
`LoadTomogram` (class in `toupy.io.dataio`), 12

## M

mask\_all() (*toupy.restoration.GUI\_tracker.PhaseTracker* method), 34

mask\_borders() (*in module toupy.utils.array\_utils*), 54

match() (*toupy.utils.funcutils.switch method*), 59

memmap\_volfile() (*in module toupy.io.filesrw*), 16

metadatafilewcard() (*toupy.io.dataio.PathName* method), 13

mod\_iradon() (*in module toupy.tomo.iradon*), 46

mod\_iradonSilx() (*in module toupy.tomo.iradon*), 47

model\_erf() (*in module toupy.utils.fit\_utils*), 57

model\_tanh() (*in module toupy.utils.fit\_utils*), 58

module

- toupy.io, 9
- toupy.io.dataio, 9
- toupy.io.filesrw, 15
- toupy.io.h5chunk\_shape\_3D, 20
- toupy.registration, 23
- toupy.registration.registration, 23
- toupy.registration.shift, 27
- toupy.resolution, 29
- toupy.resolution.FSC, 29
- toupy.resolution.FSCtools, 30
- toupy.restoration, 33
- toupy.restoration.derivativetools, 35
- toupy.restoration.GUI\_tracker, 33
- toupy.restoration.ramptools, 37
- toupy.restoration.roipoly, 38
- toupy.restoration.unwraptools, 38
- toupy.restoration.vortices tools, 41
- toupy.simulation, 43
- toupy.simulation.phantom\_creator, 43
- toupy.tomo, 45
- toupy.tomo.iradon, 45
- toupy.tomo.radon, 48
- toupy.tomo.tomorecons, 48
- toupy.utils, 51
- toupy.utils.array\_utils, 52
- toupy.utils.converter\_utils, 57
- toupy.utils.FFT\_utils, 51
- toupy.utils.fit\_utils, 57
- toupy.utils.funcutils, 59
- toupy.utils.plot\_utils, 59

oneslicefordisplay() (*in module toupy.registration.registration*), 25

onscroll() (*toupy.restoration.GUI\_tracker.PhaseTracker* method), 34

## P

padarray\_bothsides() (*in module toupy.utils.array\_utils*), 54

padfft() (*in module toupy.utils.FFT\_utils*), 51

padrightside() (*in module toupy.utils.FFT\_utils*), 52

padwidthbothsides() (*in module toupy.utils.FFT\_utils*), 52

PathName (*class in toupy.io.dataio*), 13

perturbShape() (*in module toupy.io.h5chunk\_shape\_3D*), 21

phantom() (*in module toupy.simulation.phantom\_creator*), 43

phaseresidues() (*in module toupy.restoration.unwraptools*), 39

phaseresiduesStack() (*in module toupy.restoration.unwraptools*), 39

phaseresiduesStack\_parallel() (*in module toupy.restoration.unwraptools*), 39

PhaseTracker (*class in toupy.restoration.GUI\_tracker*), 33

play() (*toupy.restoration.GUI\_tracker.PhaseTracker* method), 34

plot() (*toupy.resolution.FSC.FSCPlot method*), 29

plot\_checkangles() (*in module toupy.utils.plot\_utils*), 61

plotshorizontal() (*toupy.utils.plot\_utils.RegisterPlot* method), 59

plotsvertical() (*toupy.utils.plot\_utils.RegisterPlot* method), 59

pol2cart() (*in module toupy.restoration.vortices tools*), 42

polynomial1d() (*in module toupy.utils.array\_utils*), 54

probe2HSV() (*toupy.utils.plot\_utils.ShowProjections* static method), 60

progbar() (*in module toupy.utils.funcutils*), 59

projector() (*in module toupy.tomo.radon*), 48

projectpoly1d() (*in module toupy.utils.array\_utils*), 55

## R

nextpow2() (*in module toupy.utils.FFT\_utils*), 51

nextpoweroftwo() (*in module toupy.utils.FFT\_utils*), 51

normalize\_array() (*in module toupy.utils.array\_utils*), 54

numVals() (*in module toupy.io.h5chunk\_shape\_3D*), 21

nyquist() (*toupy.resolution.FSC.FourierShellCorr* method), 30

onclose() (*toupy.restoration.GUI\_tracker.PhaseTracker* method), 34

radonSilx() (*in module toupy.tomo.radon*), 48

radtap() (*in module toupy.utils.array\_utils*), 55

read\_cxi() (*in module toupy.io.filesrw*), 16

read\_edf() (*in module toupy.io.filesrw*), 17

read\_ptyr() (*in module toupy.io.filesrw*), 17

read\_recon() (*in module toupy.io.filesrw*), 17

read\_theta\_raw() (*in module toupy.io.filesrw*), 18

read\_theta\_recon() (*in module toupy.io.filesrw*), 18

read\_tiff() (*in module toupy.io.filesrw*), 18

read\_tiff\_info() (*in module toupy.io.filesrw*), 19

read\_volfile() (*in module toupy.io.filesrw*), 19

## O

`reconsSART()` (in module `toupy.tomo.iradon`), 47  
`refine_horizontalalignment()` (in module `toupy.registration.registration`), 25  
`register_2Darrays()` (in module `toupy.registration.registration`), 25  
`RegisterPlot` (class in `toupy.utils.plot_utils`), 59  
`remove_all_mask()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`remove_extraprojs()` (in module `toupy.io.dataio`), 14  
`remove_mask()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`remove_ramp()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`remove_rampall()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`replace_bad()` (in module `toupy.utils.array_utils`), 55  
`residuals_erf()` (in module `toupy.utils.fit_utils`), 58  
`residuals_tanh()` (in module `toupy.utils.fit_utils`), 58  
`results_datapath()` (`toupy.io.dataio.PathName` method), 13  
`results_folder()` (`toupy.io.dataio.PathName` method), 13  
`ringthickness()` (`toupy.resolution.FSC.FourierShellCorrelation` method), 30  
`rmair()` (in module `toupy.restoration.ramptools`), 37  
`rmlinearphase()` (in module `toupy.restoration.ramptools`), 37  
`rmphaseramp()` (in module `toupy.restoration.ramptools`), 37  
`rmvortices_object()` (in module `toupy.restoration.vortices_tools`), 42  
`rmvortices_probe()` (in module `toupy.restoration.vortices_tools`), 42  
`roipoly` (class in `toupy.restoration.roipoly`), 38  
`round_to_even()` (in module `toupy.utils.array_utils`), 55

## S

`save()` (`toupy.io.dataio.SaveData` class method), 13  
`save()` (`toupy.io.dataio.SaveTomogram` class method), 14  
`save_masks()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`save_vol_to_h5()` (`toupy.io.dataio.SaveTomogram` class method), 14  
`savecheck()` (`toupy.io.dataio.SaveData` method), 14  
`savecheck()` (`toupy.io.dataio.SaveTomogram` method), 14  
`SaveData` (class in `toupy.io.dataio`), 13  
`saveFSC()` (`toupy.io.dataio.SaveData` class method), 13  
`savemasks()` (`toupy.io.dataio.SaveData` class method), 14  
`SaveTomogram` (class in `toupy.io.dataio`), 14  
`search_projections()` (`toupy.io.dataio.PathName` method), 13  
`sharpening_image()` (in module `toupy.utils.array_utils`), 55  
`shift_fft()` (`toupy.registration.shift.ShiftFunc` method), 27  
`shift_linear()` (`toupy.registration.shift.ShiftFunc` method), 28  
`shift_linear_wrap()` (`toupy.registration.shift.ShiftFunc` method), 28  
`ShiftFunc` (class in `toupy.registration.shift`), 27  
`show_linearphase()` (in module `toupy.utils.plot_utils`), 61  
`show_projections()` (`toupy.utils.plot_utils.ShowProjections` method), 60  
`ShowProjections` (class in `toupy.utils.plot_utils`), 60  
`smooth1d()` (in module `toupy.utils.array_utils`), 56  
`smooth2d()` (in module `toupy.utils.array_utils`), 56  
`smooth_image()` (in module `toupy.utils.array_utils`), 57  
`sort_array()` (in module `toupy.utils.array_utils`), 57  
`split_dataset()` (in module `toupy.resolution.FSCtools`), 31  
`submit()` (`toupy.restoration.GUI_tracker.PhaseTracker` method), 34  
`switch` (class in `toupy.utils.funcutils`), 59

## T

`tiff_folderpath()` (`toupy.io.dataio.SaveTomogram` method), 14  
`tomo_recons()` (in module `toupy.tomo.tomorecons`), 49  
`tomoconsistency_multiple()` (in module `toupy.registration.registration`), 26  
`toupy.io` module, 9  
`toupy.io.dataio` module, 9  
`toupy.io.filesrw` module, 15  
`toupy.io.h5chunk_shape_3D` module, 20  
`toupy.registration` module, 23  
`toupy.registration.registration` module, 23  
`toupy.registration.shift` module, 27  
`toupy.resolution` module, 29  
`toupy.resolution.FSC` module, 29  
`toupy.resolution.FSCtools` module, 30  
`toupy.restoration` module, 33  
`toupy.restoration.derivativetools` module, 35

toupy.restoration.GUI\_tracker  
     module, 33  
 toupy.restoration.ramptools  
     module, 37  
 toupy.restoration.roipoly  
     module, 38  
 toupy.restoration.unwraptools  
     module, 38  
 toupy.restoration.vortices tools  
     module, 41  
 toupy.simulation  
     module, 43  
 toupy.simulation.phantom\_creator  
     module, 43  
 toupy.tomo  
     module, 45  
 toupy.tomo.iradon  
     module, 45  
 toupy.tomo.radon  
     module, 48  
 toupy.tomo.tomorecons  
     module, 48  
 toupy.utils  
     module, 51  
 toupy.utils.array\_utils  
     module, 52  
 toupy.utils.converter\_utils  
     module, 57  
 toupy.utils.FFT\_utils  
     module, 51  
 toupy.utils.fit\_utils  
     module, 57  
 toupy.utils.funcutils  
     module, 59  
 toupy.utils.plot\_utils  
     module, 59  
 transverse\_apodization()  
     (*toupy.resolution.FSC.FourierShellCorr*  
     *method*), 30

## U

unwrapping\_all() (*toupy.restoration.GUI\_tracker.PhaseTracker*  
     *method*), 34  
 unwrapping\_phase() (in module  
     *toupy.restoration.unwraptools*), 40  
 unwrapping\_phase()  
     (*toupy.restoration.GUI\_tracker.PhaseTracker*  
     *method*), 34  
 up() (*toupy.restoration.GUI\_tracker.PhaseTracker*  
     *method*), 34  
 update() (*toupy.restoration.GUI\_tracker.PhaseTracker*  
     *method*), 34  
 update\_show() (*toupy.utils.plot\_utils.ShowProjections*  
     *method*), 60  
 updatehorizontal()  
     (*toupy.utils.plot\_utils.RegisterPlot method*),  
     59

updatevertical() (*toupy.utils.plot\_utils.RegisterPlot*  
     *method*), 60

## V

vertical\_fluctuations() (in module  
     *toupy.registration.registration*), 26  
 vertical\_shift() (in module  
     *toupy.registration.registration*), 26

## W

wrap() (in module *toupy.restoration.unwraptools*), 40  
 wraptopi() (in module  
     *toupy.restoration.unwraptools*), 40  
 write\_edf() (in module *toupy.io.filesrw*), 19  
 write\_paramsh5() (in module *toupy.io.filesrw*), 19  
 write\_tiff() (in module *toupy.io.filesrw*), 20  
 write\_tiffmetadata() (in module *toupy.io.filesrw*),  
     20