Algotom Documentation

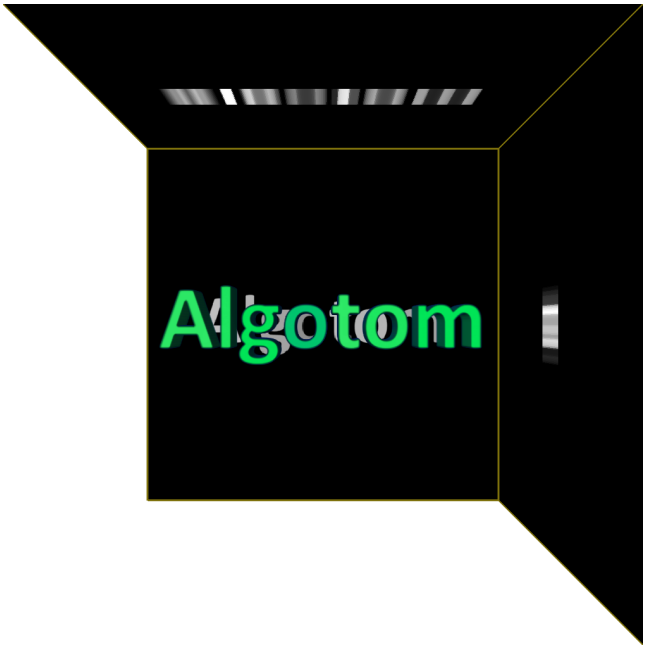
Release 1.0.3

Diamond Light Source, UK

Jul 08, 2021

CONTENTS

Data processing (**ALGO**)rithms for (**TOM**)ography



Algotom is a Python package implementing methods for processing tomographic data acquired by non-standard scanning techniques such as grid scans, helical scans, half-acquisition scans, or their combinations. Certainly, Algotom can also be used for standard scans. The software includes methods in a full pipeline of data processing: reading-writing data, pre-processing, tomographic reconstruction, post-processing, and data simulation. Many utility methods are provided to help users quickly develop prototype-methods or build a pipeline for processing their own data.

The software is made available for [A1]. Selected answers to technical questions of anonymous reviewers about methods in the paper is here.

. . . Algotom development was started at the I12-JEEP beamline in 2014 as Python codes to process data acquired by the beamline's large field-of-view (FOV) detector, which uses two imaging sensors to cover a rectangular FOV. Images from these cameras must be stitched before tomographic reconstruction can take place. Data processing methods for improving the quality of tomographic data; removing artifacts caused by imperfections of hardware components; making use the beamline capabilities; processing data acquired by non-traditional scanning techniques; and automating data processing pipeline have been actively developed at I12 over the years. These methods have been used internally by I12's users and refined further for publication and sharing with the research community through open-source software such as Tomopy and Savu . . .

. . . In contrast to Savu and Tomopy which are optimized for speed, Algotom is a package of data processing algorithms and tools which are designed to be easy-to-use and easy-to-deploy prototype methods. The development of Algotom has focused on pre-processing methods which work in the sinogram space to reduce computational cost. Methods working in the projection space such as phase filter, distortion correction, or rotation correction have been adapted to work in the sinogram space. . .

CHAPTER

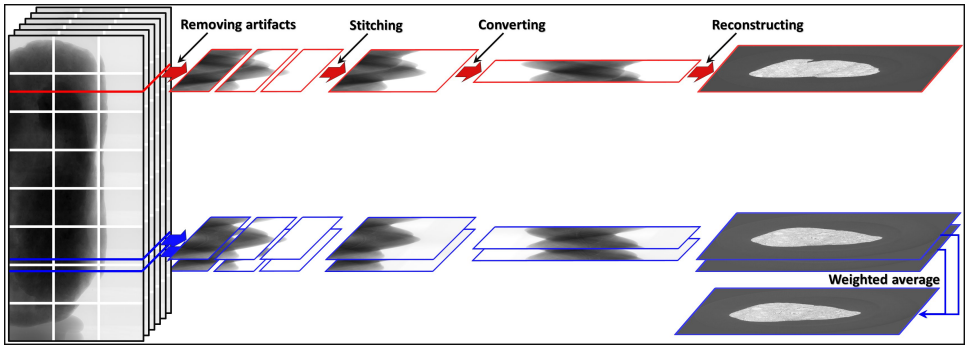
ONE

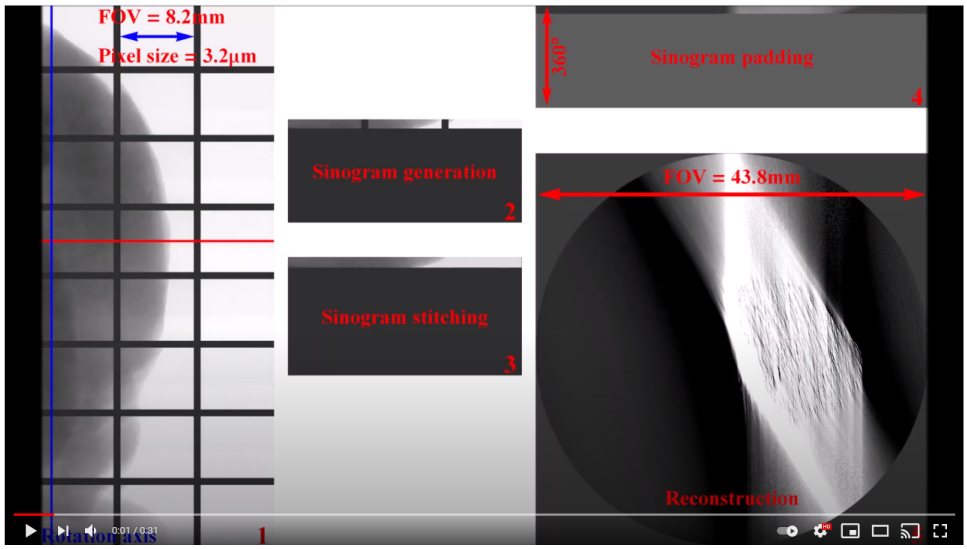
CONTENT

1.1 Features

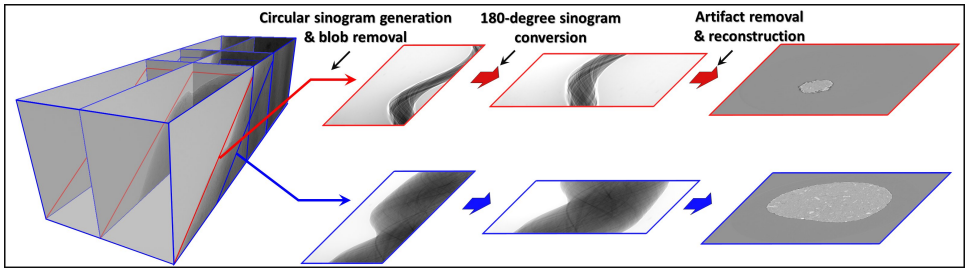
Algotom is a lightweight package. The software is built on top of a few core Python libraries to ensure its ease-of-installation. Methods distributed in Algotom have been developed and tested at a synchrotron beamline where massive datasets are produced; image features can change significantly between experiments depending on X-ray energy and sample types which can be biological, medical, material science, or geological in origin. Users often don't have sufficient experience with image processing methods to know how to properly tune parameters. All these factors drive the methods developed to be easy-to-use, robust, and practical. Some featuring methods in Algotom are as follows:

- Methods for processing grid scans (or tiled scans) with the offset rotation-axis to multiply double the field-of-view (FOV) of a parallel-beam tomography system.

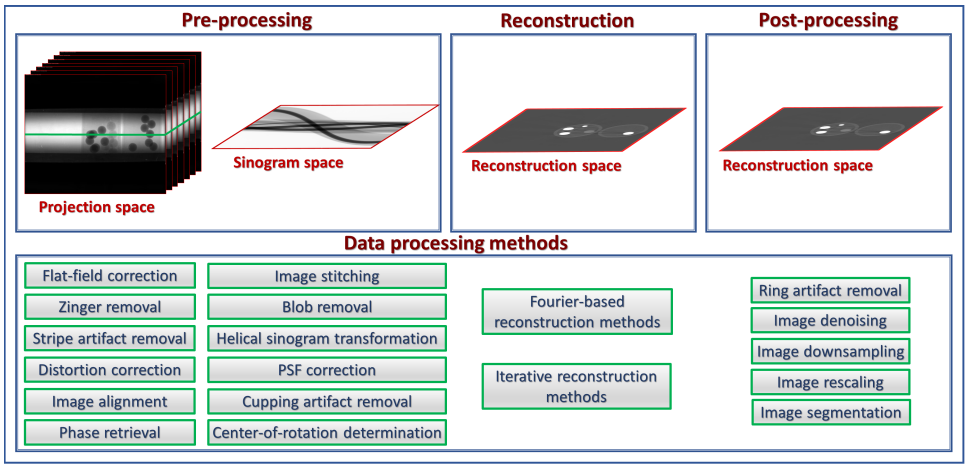




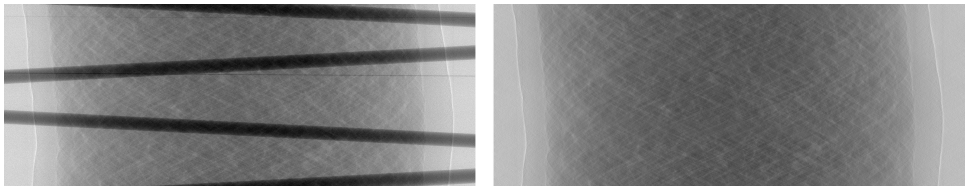
- Methods for processing helical scans (with/without the offset rotation-axis).



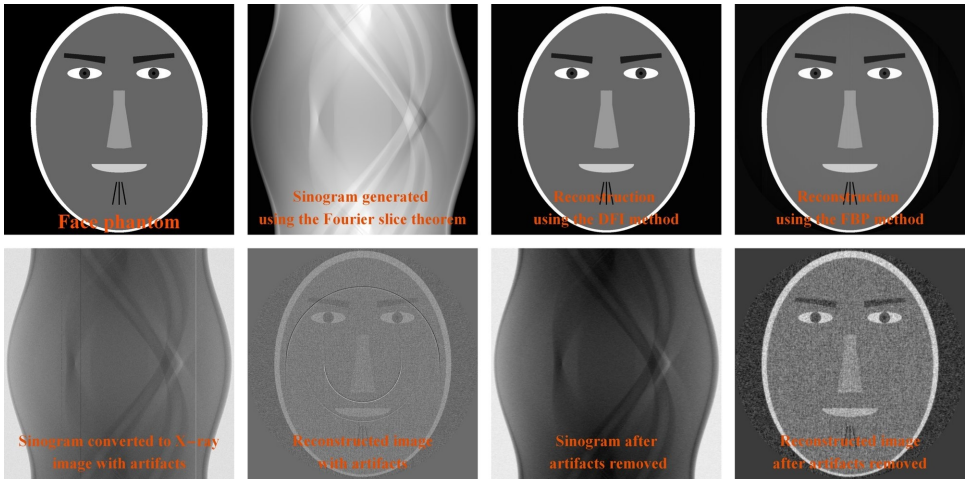
- Methods for determining the center-of-rotation (COR) and auto-stitching images in half-acquisition scans (360-degree acquisition with the offset COR).
- Methods in a full data processing pipeline: reading-writing data, pre-processing, tomographic reconstruction, and post-processing.



- Some practical methods developed and implemented for the package: zinger removal, tilted sinogram generation, sinogram distortion correction, beam hardening correction, DFI (direct Fourier inversion) reconstruction, and double-wedge filter for removing sample parts larger than the FOV in a sinogram.



- Utility methods for customizing ring/stripe artifact removal methods and parallelizing computational work.
- Calibration methods for determining pixel-size in helical scans.
- Methods for generating simulation data: phantom creation, sinogram calculation based on the Fourier slice theorem, and artifact generation.



1.2 Install

Algotom is installable across operating systems (Windows, Ubuntu, Mac) and works with Python >=3.7. To install:

1.2.1 From source

Clone the Algotom from GitHub repository:

```
git clone https://github.com/algotom/algotom.git algotom
```

Download and install Miniconda software:

```
conda create -n algotom python=3.7
conda activate algotom
cd algotom
python setup.py install
```

1.2.2 Using conda

Install Miniconda as instructed above, then:

If install to an existing environment:

```
conda install -c algotom algotom
```

If install to a new environment:

```
conda create -n algotom python=3.7
conda activate algotom
conda install -c algotom algotom
```

1.2.3 Using pip

Install Miniconda as instructed above.

If install to an existing environment:

```
pip install algotom`
```

If install to a new environment:

```
conda create -n algotom python=3.7
conda activate algotom
pip install algotom
```

1.3 Usage

- Examples of how to use the package are in the examples folder of Algotom. They cover most of use-cases which users can adapt to process their own data.
- Real tomographic data for testing methods can be downloaded from zenodo.
- Methods can also be tested using simulation data as shown in "examples/example_08*.py"
- Users can use Algotom to re-process some old data collected at synchrotron facilities suffering from:
  - Various types of ring artifacts.
  - Cupping artifacts (also known as beam hardening artifacts) which are caused by using: FFT-based reconstruction methods without proper padding; polychromatic X-ray sources; or low-dynamic-range detectors to record high-dynamic-range projection-images.

Methods distributed in Algotom can run on a normal computer which enable users to process these data locally.

- There are tools and methods users can use to customize their own algorithms:
  - Methods to transform images back-and-forth between the polar coordinate system and the Cartesian coordinate system.
  - Methods to separate stripe artifacts.
  - Methods to transform back-and-forth between reconstruction images and sinogram images.

1.4 Examples

This section contains python code examples on how to use Algotom.

1.4.1 Utility

This section contains links to python code examples.

- explore_hdf_tomo_data.py
- reconstruct_360_degree_scan_with_offset_center.py
- reconstruct_few_slices_grid_scan_with_offset_center.py
- reconstruct_helical_scan_with_offset_center.py
- reconstruct_std_scan_full_size.py
- reconstruct_std_scan.py
- reconstruct_std_scan_with_distortion_correction.py
- full_reconstruction_a_grid_scan_step_01.py
- full_reconstruction_a_grid_scan_step_02.py
- full_reconstruction_a_grid_scan_step_03_downsample.py
- generate_simulation_data.py
- generate_tilted_sinogram.py
- pre_process_data_in_the_projection_space.py

1.5 API

algotom Modules:

1.5.1 algotom.io.converter

Module for converting data type: - Convert a list of tif files to a hdf/nxs file. - Extract tif images from a hdf/nxs file.

Functions:

convert_tif_to_hdf(input_path, output_path)	Convert a folder of tif files to a hdf/nxs file.
extract_tif_from_hdf(input_path, ... [, ...])	Extract tif images from a hdf/nxs file.

algotom.io.converter.**convert_tif_to_hdf**(*input_path*, *output_path*, *key_path='entry/data'*, *crop=(0, 0, 0, 0)*, *pattern=None*, ***options*)

Convert a folder of tif files to a hdf/nxs file.

Parameters

- **input_path** (*str*) – Folder path to the tif files.
- **output_path** (*str*) – Path to the hdf/nxs file.
- **key_path** (*str, optional*) – Key path to the dataset.
- **crop** (*tuple of int, optional*) – Crop the images from the edges, i.e. crop = (crop_top, crop_bottom, crop_left, crop_right).
- **pattern** (*str, optional*) – Used to find tif files with names matching the pattern.

- **options** (*dict, optional*) – Add metadata. E.g. options={"entry/angles": angles, "entry/energy": 53}.

Returns *str* – Path to the hdf/nxs file.

algotom.io.converter.**extract_tif_from_hdf**(*input_path*, *output_path*, *key_path*, *index=(0, - 1, 1)*, *axis=0*, *crop=(0, 0, 0, 0)*, *prefix='img'*)

Extract tif images from a hdf/nxs file.

Parameters

- **input_path** (*str*) – Path to the hdf/nxs file.
- **output_path** (*str*) – Output folder.
- **key_path** (*str*) – Key path to the dataset in the hdf/nxs file.
- **index** (*tuple of int or int.*) – Indices of extracted images. A tuple corresponds to (start,stop,step).
- **axis** (*int*) – Axis which the images are extracted.
- **crop** (*tuple of int, optional*) – Crop the images from the edges, i.e. crop = (crop_top, crop_bottom, crop_left, crop_right).
- **prefix** (*str, optional*) – Prefix of names of tif files.

Returns *str* – Folder path to the tif files.

1.5.2 algotom.io.loadersaver

Module for I/O tasks: - Load data from an image file (tif, png, jpeg) or a hdf/nxs file. - Get dataset information in a hdf/nxs file. - Search for datasets in a hdf/nxs file. - Save a 2D array as a tif image or 2D, 3D array to a hdf/nxs file. - Search file names, make a file/folder name. - Load distortion coefficients from a txt file.

Functions:

load_image(file_path)	Load data from an image.
get_hdf_information(file_path)	Get information of datasets in a hdf/nxs file.
find_hdf_key(file_path, pattern)	Find datasets matching the pattern in a hdf/nxs file.
load_hdf(file_path, key_path)	Load a hdf/nexus dataset as an object.
make_folder(file_path)	Create a folder if not exist.
make_file_name(file_path)	Create a new file name to avoid overwriting.
make_folder_name(folder_path[, name_prefix])	Create a new folder name to avoid overwriting.
find_file(path)	Search file
save_image(file_path, mat[, overwrite])	Save a 2D array to an image.
open_hdf_stream(file_path, data_shape[, ...])	Write an array to a hdf/nxs file with options to add metadata.
load_distortion_coefficient(file_path)	Load distortion coefficients from a text file.
save_distortion_coefficient(file_path, ...)	Write distortion coefficients to a text file.

algotom.io.loadersaver.**find_file**(*path*)

Search file

Parameters **path** (*str*) – Path and pattern to find files.

Returns *str or list of str* – List of files.

algotom.io.loadersaver.**find_hdf_key**(*file_path*, *pattern*)
Find datasets matching the pattern in a hdf/nxs file.

Parameters

- **file_path** (*str*) – Path to the file.
- **pattern** (*str*) – Pattern to find the full names of the datasets.

Returns

- **list_key** (*str*) – Keys to the datasets.
- **list_shape** (*tuple of int*) – Shapes of the datasets.
- **list_type** (*str*) – Types of the datasets.

algotom.io.loadersaver.**get_hdf_information**(*file_path*)
Get information of datasets in a hdf/nxs file.

Parameters **file_path** (*str*) – Path to the file.

Returns

- **list_key** (*str*) – Keys to the datasets.
- **list_shape** (*tuple of int*) – Shapes of the datasets.
- **list_type** (*str*) – Types of the datasets.

algotom.io.loadersaver.**load_distortion_coefficient**(*file_path*)
Load distortion coefficients from a text file.

Parameters **file_path** (*str*) – Path to the file

Returns *tuple of float and list* – Tuple of (xcenter, ycenter, list_fact).

algotom.io.loadersaver.**load_hdf**(*file_path*, *key_path*)
Load a hdf/nexus dataset as an object.

Parameters

- **file_path** (*str*) – Path to the file.
- **key_path** (*str*) – Key path to the dataset.

Returns *object* – hdf/nxs object.

algotom.io.loadersaver.**load_image**(*file_path*)
Load data from an image.

Parameters **file_path** (*str*) – Path to the file.

Returns *float* – 2D array.

algotom.io.loadersaver.**make_file_name**(*file_path*)
Create a new file name to avoid overwriting.

Parameters **file_path** (*str*)

Returns *str* – Updated file path.

algotom.io.loadersaver.**make_folder**(*file_path*)
Create a folder if not exist.

Parameters **file_path** (*str*)

algotom.io.loadersaver.**make_folder_name**(*folder_path*, *name_prefix='Output'*)
Create a new folder name to avoid overwriting. E.g: Output_00001, Output_00002…

Parameters

- **folder_path** (*str*) – Path to the parent folder.
- **name_prefix** (*str*) – Name prefix

Returns *str* – Name of the folder.

algotom.io.loadersaver.**open_hdf_stream**(*file_path*, *data_shape*, *key_path='entry/data'*, *data_type='float32'*, *overwrite=True*, ***options*)

Write an array to a hdf/nxs file with options to add metadata.

Parameters

- **file_path** (*str*) – Path to the file.
- **data_shape** (*tuple of int*) – Shape of the data.
- **key_path** (*str*) – Key path to the dataset.
- **data_type** (*str*) – Type of data.
- **overwrite** (*bool*) – Overwrite the existing file if True.
- **options** (*dict, optional*) – Add metadata. E.g. options={"entry/angles": angles, "entry/energy": 53}.

Returns *object* – hdf object.

algotom.io.loadersaver.**save_distortion_coefficient**(*file_path*, *xcenter*, *ycenter*, *list_fact*, *overwrite=True*)

Write distortion coefficients to a text file.

Parameters

- **file_path** (*str*) – Path to the file.
- **xcenter** (*float*) – Center of distortion in x-direction.
- **ycenter** (*float*) – Center of distortion in y-direction.
- **list_fact** (*float*) – 1D array. Coefficients of the polynomial fit.
- **overwrite** (*bool*) – Overwrite an existing file if True.

Returns *str* – Updated file path.

algotom.io.loadersaver.**save_image**(*file_path*, *mat*, *overwrite=True*)

Save a 2D array to an image.

Parameters

- **file_path** (*str*) – Path to the file.
- **mat** (*int or float*) – 2D array.
- **overwrite** (*bool*) – Overwrite an existing file if True.

Returns *str* – Updated file path.

1.5.3 algotom.post.postprocessing

Module of methods in the postprocessing stage: - Get statistical information of reconstructed images or a dataset. - Downsample 2D, 3D array, or a dataset. - Rescale 2D, 3D array or a dataset to 8-bit or 16-bit data-type. - Removing ring artifacts in a reconstructed image by transform back and forth between the polar coordinates and the Cartesian coordinates.

Functions:

get_statical_information(mat[, percentile, ...])	Get statical information of an image.
get_statical_information_dataset(input_[, ...])	Get statical information of a dataset.
downsample(mat, cell_size[, method])	Downsample an image.
downsample_dataset(input_, output, cell_size)	Downsample a dataset.
rescale(mat[, nbit, minmax])	Rescale a 32-bit array to 16-bit/8-bit data.
rescale_dataset(input_, output[, nbit, ...])	Rescale a dataset to 8-bit or 16-bit data-type.
remove_ring_based_fft(mat[, u, n, v, sort])	Remove ring artifacts in the reconstructed image by combining the polar transform and the fft-based method.
remove_ring_based_wavelet_fft(mat[, level, ...])	Remove ring artifacts in a reconstructed image by combining the polar transform and the wavelet-fft-based method (Ref.

algotom.post.postprocessing.**downsample**(*mat*, *cell_size*, *method='mean'*)

Downsample an image.

Parameters

- **mat** (*array_like*) – 2D array.
- **cell_size** (*int or tuple of int*) – Window size along axes used for grouping pixels.
- **method** (*{"mean", "median", "max", "min"}*) – Downsampling method.

Returns *array_like* – Downsampled image.

algotom.post.postprocessing.**downsample_dataset**(*input_*, *output*, *cell_size*, *method='mean'*, *key_path=None*)

Downsample a dataset. This can be a folder of tif files, a hdf file, or a 3D array.

Parameters

- **input_** (*str, array_like*) – It can be a folder path to tif files, a hdf file, or 3D array.
- **output** (*str, None*) – It can be a folder path, a hdf file path, or None (memory consuming).
- **cell_size** (*int or tuple of int*) – Window size along axes used for grouping pixels.
- **method** (*{"mean", "median", "max", "min"}*) – Downsampling method.
- **key_path** (*str, optional*) – Key path to the dataset if the input is the hdf file.

Returns *array_like or None* – If output is None, returning an 3D array.

algotom.post.postprocessing.**get_statical_information**(*mat*, *percentile=(5, 95)*, *denoise=False*)

Get statical information of an image.

Parameters

- **mat** (*array_like*) – 2D array. Projection image, sinogram image, or reconstructed image.

- **percentile** (*tuple of floats*) – Tuple of (min_percentile, max_percentile) to compute. Must be between 0 and 100 inclusive.
- **denoise** (*bool, optional*) – Enable/disable denoising before extracting statistical information.

Returns

- **gmin** (*float*) – The minimum value of the data array.
- **gmax** (*float*) – The maximum value of the data array.
- **min_percent** (*float*) – The first computed percentile of the data array.
- **max_percent** (*tuple of floats*) – The last computed percentile of the data array.
- **mean** (*float*) – The mean of the data array.
- **median** (*float*) – The median of the data array.
- **variance** (*float*) – The variance of the data array.

algotom.post.postprocessing.**get_statical_information_dataset**(*input_*, *percentile=(5, 95)*, *skip=5*, *denoise=False*, *key_path=None*)

Get statical information of a dataset. This can be a folder of tif files, a hdf file, or a 3D array.

Parameters

- **input_** (*str, hdf file, or array_like*) – It can be a folder path to tif files, a hdf file, or a 3D array.
- **percentile** (*tuple of floats*) – Tuple of (min_percentile, max_percentile) to compute. Must be between 0 and 100 inclusive.
- **skip** (*int*) – Skipping step of reading input.
- **denoise** (*bool, optional*) – Enable/disable denoising before extracting statistical information.
- **key_path** (*str, optional*) – Key path to the dataset if the input is the hdf file.

Returns

- **gmin** (*float*) – The global minimum value of the data array.
- **gmax** (*float*) – The global maximum value of the data array.
- **min_percent** (*float*) – The global min of the first computed percentile of the data array.
- **max_percent** (*tuple of floats*) – The global min of the last computed percentile of the data array.
- **mean** (*float*) – The mean of the data array.
- **median** (*float*) – The median of the data array.
- **variance** (*float*) – The mean of the variance of the data array.

algotom.post.postprocessing.**remove_ring_based_fft**(*mat*, *u=20*, *n=8*, *v=1*, *sort=False*)

Remove ring artifacts in the reconstructed image by combining the polar transform and the fft-based method.

Parameters

- **mat** (*array_like*) – Square array. Reconstructed image
- **u** (*int*) – Cutoff frequency.
- **n** (*int*) – Filter order.
- **v** (*int*) – Number of rows (* 2) to be applied the filter.

- **sort** (*bool, optional*) – Apply sorting (Ref. [2]) if True.

Returns *array_like* – Ring-removed image.

References

algotom.post.postprocessing.**remove_ring_based_wavelet_fft**(*mat*, *level=5*, *size=1*, *wavelet_name='db9'*, *sort=False*)

Remove ring artifacts in a reconstructed image by combining the polar transform and the wavelet-fft-based method (Ref. [1]).

Parameters

- **mat** (*array_like*) – Square array. Reconstructed image
- **level** (*int*) – Wavelet decomposition level.
- **size** (*int*) – Damping parameter. Larger is stronger.
- **wavelet_name** (*str*) – Name of a wavelet. Search pywavelets API for a full list.
- **sort** (*bool, optional*) – Apply sorting (Ref. [2]) if True.

Returns *array_like* – Ring-removed image.

References

algotom.post.postprocessing.**rescale**(*mat*, *nbit=16*, *minmax=None*)

Rescale a 32-bit array to 16-bit/8-bit data.

Parameters

- **mat** (*array_like*)
- **nbit** (*{8,16}*) – Rescaled data-type: 8-bit or 16-bit.
- **minmax** (*tuple of float, or None*) – Minimum and maximum values used for rescaling.

Returns *array_like* – Rescaled array.

algotom.post.postprocessing.**rescale_dataset**(*input_*, *output*, *nbit=16*, *minmax=None*, *skip=None*, *key_path=None*)

Rescale a dataset to 8-bit or 16-bit data-type. The dataset can be a folder of tif files, a hdf file, or a 3D array.

Parameters

- **input_** (*str, array_like*) – It can be a folder path to tif files, a hdf file, or 3D array.
- **output** (*str, None*) – It can be a folder path, a hdf file path, or None (memory consuming).
- **nbit** (*{8,16}*) – Rescaled data-type: 8-bit or 16-bit.
- **minmax** (*tuple of float, or None*) – Minimum and maximum values used for rescaling. They are calculated if None is given.
- **skip** (*int or None*) – Skipping step of reading input used for getting statistical information.
- **key_path** (*str, optional*) – Key path to the dataset if the input is the hdf file.

Returns *array_like or None* – If output is None, returning an 3D array.

1.5.4 algotom.prep.calculation

Module of calculation methods in the preprocessing stage: - Calculating the center-of-rotation (COR) in a 180-degree scan using a

sinogram.

- Determining the overlap-side and overlap-area between images.
- Calculating the COR in a half-acquisition scan (360-degree scan with offset COR).
- Using the similar technique as above to calculate the COR in a 180-degree scan from two projections.
- Determining the relative translations between images using phase-correlation technique.
- Calculating the COR in a 180-degree scan using phase-correlation technique.

Functions:

—

algotom.prep.calculation.**calculate_center_metric**(*center*, *sino_180*, *sino_flip*, *sino_comp*, *mask*)
Calculate a metric of an estimated center-of-rotation.

Parameters

- **center** (*float*) – Estimated center.
- **sino_180** (*array_like*) – 2D array. 180-degree sinogram.
- **sino_flip** (*array_like*) – 2D array. Flip the 180-degree sinogram in the left/right direction.
- **sino_comp** (*array_like*) – 2D array. Used to fill the gap left by image shifting.
- **mask** (*array_like*) – 2D array. Used to select coefficients in the double-wedge region.

Returns *float* – Metric.

algotom.prep.calculation.**calculate_curvature**(*list_metric*)
Calculate the curvature of a fitted curve going through the minimum value of a metric list.

Parameters **list_metric** (*array_like*) – 1D array. List of metrics.

Returns

- **curvature** (*float*) – Quadratic coefficient of the parabola fitting.
- **min_pos** (*float*) – Position of the minimum value with sub-pixel accuracy.

algotom.prep.calculation.**calculate_maximum_index**(*y_start*, *y_stop*, *pitch*, *pixel_size*, *scan_type*)
Calculate the maximum index of a reconstructable slice in a helical scan.

Parameters

- **y_start** (*float*) – Y-position of the stage at the beginning of the scan.
- **y_stop** (*float*) – Y-position of the stage at the end of the scan.
- **pitch** (*float*) – The distance which the y-stage is translated in one full rotation.
- **pixel_size** (*float*) – Pixel size. The unit must be the same as y-position.
- **scan_type** (*{"180", "360"}*) – One of two options: "180" for generating a 180-degree sinogram or "360" for generating a 360-degree sinogram.

Returns *int* – Maximum index of reconstructable slices.

algotom.prep.calculation.**calculate_reconstructable_height**(*y_start*, *y_stop*, *pitch*, *scan_type*)
Calculate reconstructable height in a helical scan.

Parameters

- **y_start** (*float*) – Y-position of the stage at the beginning of the scan.
- **y_stop** (*float*) – Y-position of the stage at the end of the scan.
- **pitch** (*float*) – The distance which the y-stage is translated in one full rotation.
- **scan_type** (*{"180", "360"}*) – One of two options: "180" for generating a 180-degree sinogram or "360" for generating a 360-degree sinogram.

Returns

- **y_s** (*float*) – Starting point of the reconstructable height.
- **y_e** (*float*) – End point of the reconstructable height.

algotom.prep.calculation.**coarse_search_cor**(*sino_180*, *start*, *stop*, *ratio=0.5*, *denoise=True*, *ncore=None*)
Find the center-of-rotation (COR) using integer shifting.

Parameters

- **sino_180** (*array_like*) – 2D array. 180-degree sinogram.
- **start** (*int*) – Starting point for searching COR.
- **stop** (*int*) – Ending point for searching COR.
- **ratio** (*float*) – Ratio between a sample and the width of the sinogram.
- **denoise** (*bool, optional*) – Apply a smoothing filter.
- **ncore** (*int or None*) – Number of cores used for computing. Automatically selected if None.

Returns *float* – Center of rotation.

algotom.prep.calculation.**complex_gradient**(*mat*)
Return complex gradient of a 2D array.

algotom.prep.calculation.**correlation_metric**(*mat1*, *mat2*)
Calculate the correlation metric. Smaller metric corresponds to better correlation.

Parameters

- **mat1** (*array_like*)
- **mat2** (*array_like*)

Returns *float* – Correlation metric.

algotom.prep.calculation.**downsample_cor**(*image*, *dsp_fact0*, *dsp_fact1*)
Downsample an image by averaging.

Parameters

- **image** (*array_like*) – 2D array.
- **dsp_fact0** (*int*) – Downsampling factor along axis 0.
- **dsp_fact1** (*int*) – Downsampling factor along axis 1.

Returns *array_like* – 2D array. Downsampled image.

algotom.prep.calculation.**find_center_360**(*sino_360*, *win_width*, *side=None*, *denoise=True*, *norm=False*, *use_overlap=False*)

Find the center-of-rotation (COR) in a 360-degree scan with offset COR use the method presented in Ref. [1].

Parameters

- **sino_360** (*array_like*) – 2D array. 360-degree sinogram.
- **win_width** (*int*) – Window width used for finding the overlap area.
- **side** (*{None, 0, 1}, optional*) – Overlap size. Only there options: None, 0, or 1. "None" corresponding to fully automated determination. "0" corresponding to the left side. "1" corresponding to the right side.
- **denoise** (*bool, optional*) – Apply the Gaussian filter if True.
- **norm** (*bool, optional*) – Apply the normalization if True.
- **use_overlap** (*bool, optional*) – Use the combination of images in the overlap area for calculating correlation coefficients if True.

Returns

- **cor** (*float*) – Center-of-rotation.
- **overlap** (*float*) – Width of the overlap area between two halves of the sinogram.
- **side** (*int*) – Overlap side between two halves of the sinogram.
- **overlap_position** (*float*) – Position of the window in the first image giving the best correlation metric.

References

algotom.prep.calculation.**find_center_based_phase_correlation**(*mat1*, *mat2*, *flip=True*, *gradient=True*)

Find the center-of-rotation (COR) using projection images at 0-degree and 180-degree.

Parameters

- **mat1** (*array_like*) – 2D array. Projection image at 0-degree.
- **mat2** (*array_like*) – 2D array. Projection image at 180-degree.
- **flip** (*bool, optional*) – Flip the 180-degree projection in the left-right direction if True.
- **gradient** (*bool, optional*) – Use the complex gradient of the input image for calculation.

Returns **cor** (*float*) – Center-of-rotation.

algotom.prep.calculation.**find_center_projection**(*mat1*, *mat2*, *flip=True*, *chunk_height=None*, *start_row=None*, *denoise=True*, *norm=False*, *use_overlap=False*)

Find the center-of-rotation (COR) using projection images at 0-degree and 180-degree based on a method in Ref. [1].

Parameters

- **mat1** (*array_like*) – 2D array. Projection image at 0-degree.
- **mat2** (*array_like*) – 2D array. Projection image at 180-degree.
- **flip** (*bool, optional*) – Flip the 180-degree projection in the left-right direction if True.

- **chunk_height** (*int or float, optional*) – Height of the sub-area of projection images. If a float is given, it must be in the range of [0.0, 1.0].
- **start_row** (*int, optional*) – Starting row used to extract the sub-area.
- **denoise** (*bool, optional*) – Apply the Gaussian filter if True.
- **norm** (*bool, optional*) – Apply the normalization if True.
- **use_overlap** (*bool, optional*) – Use the combination of images in the overlap area for calculating correlation coefficients if True.

Returns **cor** (*float*) – Center-of-rotation.

References

algotom.prep.calculation.**find_center_vo**(*sino_180*, *start=None*, *stop=None*, *step=0.25*, *radius=4*, *ratio=0.5*, *dsp=True*, *ncore=None*)

Find the center-of-rotation using the method described in Ref. [1].

Parameters

- **sino_180** (*array_like*) – 2D array. 180-degree sinogram.
- **start** (*float*) – Starting point for searching CoR.
- **stop** (*float*) – Ending point for searching CoR.
- **step** (*float*) – Sub-pixel accuracy of estimated CoR.
- **radius** (*float*) – Searching range with the sub-pixel step.
- **ratio** (*float*) – Ratio between the sample and the width of the sinogram.
- **dsp** (*bool*) – Enable/disable downsampling.
- **ncore** (*int or None*) – Number of cores used for computing. Automatically selected if None.

Returns *float* – Center-of-rotation.

References

algotom.prep.calculation.**find_overlap**(*mat1*, *mat2*, *win_width*, *side=None*, *denoise=True*, *norm=False*, *use_overlap=False*)

Find the overlap area and overlap side between two images (Ref. [1]) where the overlap side referring to the first image.

Parameters

- **mat1** (*array_like*) – 2D array. Projection image or sinogram image.
- **mat2** (*array_like*) – 2D array. Projection image or sinogram image.
- **win_width** (*int*) – Width of the searching window.
- **side** (*{None, 0, 1}, optional*) – Only there options: None, 0, or 1. "None" corresponding to fully automated determination. "0" corresponding to the left side. "1" corresponding to the right side.
- **denoise** (*bool, optional*) – Apply the Gaussian filter if True.
- **norm** (*bool, optional*) – Apply the normalization if True.

- **use_overlap** (*bool, optional*) – Use the combination of images in the overlap area for calculating correlation coefficients if True.

Returns

- **overlap** (*float*) – Width of the overlap area between two images.
- **side** (*int*) – Overlap side between two images.
- **overlap_position** (*float*) – Position of the window in the first image giving the best correlation metric.

References

algotom.prep.calculation.**find_overlap_multiple**(*list_mat*, *win_width*, *side=None*, *denoise=True*, *norm=False*, *use_overlap=False*)

Find the overlap-areas and overlap-sides of a list of images where the overlap side referring to the previous image.

Parameters

- **list_mat** (*list of array_like*) – List of 2D array. Projection image or sinogram image.
- **win_width** (*int*) – Width of the searching window.
- **side** (*{None, 0, 1}, optional*) – Only there options: None, 0, or 1. "None" corresponding to fully automated determination. "0" corresponding to the left side. "1" corresponding to the right side.
- **denoise** (*bool, optional*) – Apply the Gaussian filter if True.
- **norm** (*bool, optional*) – Apply the normalization if True.
- **use_overlap** (*bool, optional*) – Use the combination of images in the overlap area for calculating correlation coefficients if True.

Returns **list_overlap** (*list of tuple of floats*) – List of [overlap, side, overlap_position]. overlap : Width of the overlap area between two images. side : Overlap side between two images. overlap_position : Position of the window in the first image giving the best correlation metric.

algotom.prep.calculation.**find_shift_based_phase_correlation**(*mat1*, *mat2*, *gradient=True*)

Find relative translation in x and y direction between images with haft-pixel accuracy (Ref. [1]).

Parameters

- **mat1** (*array_like*) – 2D array. Projection image or sinogram image.
- **mat2** (*array_like*) – 2D array. Projection image or sinogram image.
- **gradient** (*bool, optional*) – Use the complex gradient of the input image for calculation.

Returns

- **ty** (*float*) – Translation in y-direction.
- **tx** (*float*) – Translation in x-direction.

References

algotom.prep.calculation.**fine_search_cor**(*sino_180*, *start*, *radius*, *step*, *ratio=0.5*, *denoise=True*, *ncore=None*)

Find the center-of-rotation (COR) using sub-pixel shifting.

Parameters

- **sino_180** (*array_like*) – 2D array. 180-degree sinogram.
- **start** (*float*) – Starting point for searching COR.
- **radius** (*float*) – Searching range: [start - radius; start + radius].
- **step** (*float*) – Searching step.
- **ratio** (*float*) – Ratio between a sample and the width of the sinogram.
- **denoise** (*bool, optional*) – Apply a smoothing filter.
- **ncore** (*int or None*) – Number of cores used for computing. Automatically selected if None.

Returns *float* – Center of rotation.

algotom.prep.calculation.**make_inverse_double_wedge_mask**(*height*, *width*, *radius*)

Generate a double-wedge binary mask using Eq. (3) in Ref. [1]. Values outside the double-wedge region correspond to 1.0.

Parameters

- **height** (*int*) – Image height.
- **width** (*int*) – Image width.
- **radius** (*int*) – Radius of an object, in pixel unit.

Returns *array_like* – 2D binary mask.

References

algotom.prep.calculation.**search_overlap**(*mat1*, *mat2*, *win_width*, *side*, *denoise=True*, *norm=False*, *use_overlap=False*)

Calculate the correlation metrics between a rectangular region, defined by the window width, on the utmost left/right side of image 2 and the same size region in image 1 where the region is slided across image 1.

Parameters

- **mat1** (*array_like*) – 2D array. Projection image or sinogram image.
- **mat2** (*array_like*) – 2D array. Projection image or sinogram image.
- **win_width** (*int*) – Width of the searching window.
- **side** (*{0, 1}*) – Only two options: 0 or 1. It is used to indicate the overlap side respects to image 1. "0" corresponds to the left side. "1" corresponds to the right side.
- **denoise** (*bool, optional*) – Apply the Gaussian filter if True.
- **norm** (*bool, optional*) – Apply the normalization if True.
- **use_overlap** (*bool, optional*) – Use the combination of images in the overlap area for calculating correlation coefficients if True.

Returns

- **list_metric** (*array_like*) – 1D array. List of the correlation metrics.

- **offset** (*int*) – Initial position of the searching window where the position corresponds to the center of the window.

1.5.5 algotom.prep.conversion

Module of conversion methods in the preprocessing stage: - Stitching images. - Joining images if there is no overlapping. - Converting a 360-degree sinogram with offset center-of-rotation (COR) to

a 180-degree sinogram.

- Extending a 360-degree sinogram with offset COR for direct reconstruction instead of converting it to a 180-degree sinogram.
- Converting a 180-degree sinogram to a 360-sinogram.
- Generating a sinogram from a helical data.

Functions:

—

algotom.prep.conversion.**convert_sinogram_180_to_360**(*sino_180*, *center*)
Convert a 180-degree sinogram to a 360-degree sinogram (Ref. [1]).

Parameters

- **sino_180** (*array_like*) – 2D array. 180-degree sinogram.
- **center** (*float*) – Center-of-rotation.

Returns *array_like* – 360-degree sinogram.

References

algotom.prep.conversion.**convert_sinogram_360_to_180**(*sino_360*, *cor*, *wei_mat1=None*, *wei_mat2=None*, *norm=True*, *total_width=None*)
Convert a 360-degree sinogram to a 180-degree sinogram.

Parameters

- **sino_360** (*array_like*) – 2D array. 360-degree sinogram.
- **cor** (*float or tuple of float*) – Center-of-rotation or (Overlap_area, overlap_side).
- **wei_mat1** (*array_like, optional*) – Weighting matrix used for the 1st haft of the sinogram.
- **wei_mat2** (*array_like, optional*) – Weighting matrix used for the 2nd haft of the sinogram.
- **norm** (*bool, optional*) – Enable/disable normalization before stitching.
- **total_width** (*int, optional*) – Final width of the stitched image.

Returns

- **sino_stiched** (*array_like*) – Converted sinogram.
- **cor** (*float*) – Updated center-of-rotation referred to the converted sinogram.

algotom.prep.conversion.**extend_sinogram**(*sino_360*, *cor*, *apply_log=True*)
Extend a 360-degree sinogram (with offset center-of-rotation) for later reconstruction (Ref. [1]).

Parameters

- **sino_360** (*array_like*) – 2D array. 360-degree sinogram.
- **cor** (*float or tuple of float*) – Center-of-rotation or (Overlap_area, overlap_side).
- **apply_log** (*bool, optional*) – Apply the logarithm function if True.

Returns

- **sino_pad** (*array_like*) – Extended sinogram.
- **cor** (*float*) – Updated center-of-rotation referred to the converted sinogram.

References

algotom.prep.conversion.**generate_full_sinogram_helical_scan**(*index*, *tomo_data*, *num_proj*, *pixel_size*, *y_start*, *y_stop*, *pitch*, *scan_type='180'*, *angles=None*, *flat=None*, *dark=None*, *mask=None*, *crop=(0, 0, 0, 0)*)

Generate a full sinogram from a helical scan dataset which is a hdf/nxs object (Ref. [1]). Full sinogram is all 1D projection of the same slice of a sample staying inside the field of view.

Parameters

- **index** (*int*) – Index of the sinogram.
- **tomo_data** (*hdf object.*) – 3D array.
- **num_proj** (*int*) – Number of projections per 180-degree.
- **pixel_size** (*float*) – Pixel size. The unit must be the same as y-position.
- **y_start** (*float*) – Y-position of the stage at the beginning of the scan.
- **y_stop** (*float*) – Y-position of the stage at the end of the scan.
- **pitch** (*float*) – The distance which the y-stage is translated in one full rotation.
- **scan_type** (*{"180", "360"}*) – Data acquired is the 180-degree type or 360-degree type [1].
- **angles** (*array_like, optional*) – 1D array. List of angles (degree) corresponding to acquired projections.
- **flat** (*array_like, optional*) – Flat-field image used for flat-field correction.
- **dark** (*array_like, optional*) – Dark-field image used for flat-field correction.
- **mask** (*array_like, optional*) – Used for removing streak artifacts caused by blobs in the flat-field image.
- **crop** (*tuple of int, optional*) – Used for cropping images.

Returns

- **sinogram** (*array_like*) – 2D array. Full sinogram.
- **list_angle** (*array_like*) – 1D array. List of angles corresponding to the generated sinogram.

References

algotom.prep.conversion.**generate_sinogram_helical_scan**(*index*, *tomo_data*, *num_proj*, *pixel_size*, *y_start*, *y_stop*, *pitch*, *scan_type='180'*, *angles=None*, *flat=None*, *dark=None*, *mask=None*, *crop=(0, 0, 0, 0)*)

Generate a 180-degree sinogram or a 360-degree sinogram from a helical scan dataset which is a hdf/nxs object (Ref. [1]).

Parameters

- **index** (*int*) – Index of the sinogram.
- **tomo_data** (*hdf object.*) – 3D array.
- **num_proj** (*int*) – Number of projections per 180-degree.
- **pixel_size** (*float*) – Pixel size. The unit must be the same as y-position.
- **y_start** (*float*) – Y-position of the stage at the beginning of the scan.
- **y_stop** (*float*) – Y-position of the stage at the end of the scan.
- **pitch** (*float*) – The distance which the y-stage is translated in one full rotation.
- **scan_type** (*{"180", "360"}*) – One of two options: "180" for generating a 180-degree sinogram or "360" for generating a 360-degree sinogram.
- **angles** (*array_like, optional*) – 1D array. List of angles (degree) corresponding to acquired projections.
- **flat** (*array_like, optional*) – Flat-field image used for flat-field correction.
- **dark** (*array_like, optional*) – Dark-field image used for flat-field correction.
- **mask** (*array_like, optional*) – Used for removing streak artifacts caused by blobs in the flat-field image.
- **crop** (*tuple of int, optional*) – Used for cropping images.

Returns

- **sinogram** (*array_like*) – 2D array. 180-degree sinogram or 360-degree sinogram.
- **list_angle** (*array_like*) – 1D array. List of angles corresponding to the generated sinogram.

References

algotom.prep.conversion.**join_image**(*mat1*, *mat2*, *joint_width*, *side*, *norm=True*, *total_width=None*)

Join projection images or sinogram images. This is useful for fixing the problem of non-overlap between images.

Parameters

- **mat1** (*array_like*) – 2D array. Projection image or sinogram image.
- **mat2** (*array_like*) – 2D array. Projection image or sinogram image.
- **joint_width** (*float*) – Width of the joint area between two images.
- **side** (*{0, 1}*) – Only two options: 0 or 1. It is used to indicate the overlap side respects to image 1. "0" corresponds to the left side. "1" corresponds to the right side.
- **norm** (*bool*) – Enable/disable normalization before joining.
- **total_width** (*int, optional*) – Final width of the joined image.

Returns *array_like* – Stitched image.

algotom.prep.conversion.**join_image_multiple**(*list_mat*, *list_joint*, *norm=True*, *total_width=None*)
Join list of projection images or sinogram images. This is useful for fixing the problem of non-overlap between images.

Parameters

- **list_mat** (*list of array_like*) – List of 2D array. Projection image or sinogram image.
- **list_joint** (*list of tuple of floats*) – List of [joint_width, side]. joint_width : Width of the joint area between two images. side : Overlap side between two images.
- **norm** (*bool, optional*) – Enable/disable normalization before stitching.
- **total_width** (*int, optional*) – Final width of the stitched image.

Returns *array_like* – Stitched image.

algotom.prep.conversion.**make_weight_matrix**(*mat1*, *mat2*, *overlap*, *side*)
Generate a linear-ramp weighting matrix for image stitching.

Parameters

- **mat1** (*array_like*) – 2D array. Projection image or sinogram image.
- **mat2** (*array_like*) – 2D array. Projection image or sinogram image.
- **overlap** (*int*) – Width of the overlap area between two images.
- **side** (*{0, 1}*) – Only two options: 0 or 1. It is used to indicate the overlap side respects to image 1. "0" corresponds to the left side. "1" corresponds to the right side.

algotom.prep.conversion.**stitch_image**(*mat1*, *mat2*, *overlap*, *side*, *wei_mat1=None*, *wei_mat2=None*, *norm=True*, *total_width=None*)
Stitch projection images or sinogram images using a linear ramp.

Parameters

- **mat1** (*array_like*) – 2D array. Projection image or sinogram image.
- **mat2** (*array_like*) – 2D array. Projection image or sinogram image.
- **overlap** (*float*) – Width of the overlap area between two images.
- **side** (*{0, 1}*) – Only two options: 0 or 1. It is used to indicate the overlap side respects to image 1. "0" corresponds to the left side. "1" corresponds to the right side.
- **wei_mat1** (*array_like, optional*) – Weighting matrix used for image 1.
- **wei_mat2** (*array_like, optional*) – Weighting matrix used for image 2.
- **norm** (*bool, optional*) – Enable/disable normalization before stitching.
- **total_width** (*int, optional*) – Final width of the stitched image.

Returns *array_like* – Stitched image.

algotom.prep.conversion.**stitch_image_multiple**(*list_mat*, *list_overlap*, *norm=True*, *total_width=None*)
Stitch list of projection images or sinogram images using a linear ramp.

Parameters

- **list_mat** (*list of array_like*) – List of 2D array. Projection image or sinogram image.
- **list_overlap** (*list of tuple of floats*) – List of [overlap, side]. overlap : Width of the overlap area between two images. side : Overlap side between two images.

- **norm** (*bool, optional*) – Enable/disable normalization before stitching.
- **total_width** (*int, optional*) – Final width of the stitched image.

Returns *array_like* – Stitched image.

1.5.6 algotom.prep.correction

Module of correction methods in the preprocessing stage: - Flat-field correction. - Distortion correction. - MTF deconvolution. - Tilted sinogram generation. - Tilted 1D intensity-profile generation. - Beam hardening correction.

Functions:

—

algotom.prep.correction.**beam_hardening_correction**(*mat*, *q*, *n*, *opt=True*)
Correct the grayscale values of a normalized image using a non-linear function.

Parameters

- **mat** (*array_like*) – Normalized projection image or sinogram image.
- **q** (*float*) – Positive number. Recommended range [0.005, 50].
- **n** (*float*) – Positive number. Must larger than 1.
- **opt** (*bool*) – True: Curve towards 0.0. False: Curve towards 1.0.

Returns *array_like* – Corrected image.

algotom.prep.correction.**flat_field_correction**(*proj*, *flat*, *dark*, *ratio=1.0*, *use_dark=True*, ***options*)
Do flat-field correction with options to remove zinger arifacts and/or stripe artifacts.

Parameters

- **proj** (*array_like*) – 3D or 2D array. Projection images or a sinogram image.
- **flat** (*array_like*) – 2D or 1D array. Flat-field image or a single row of it.
- **dark** (*array_like*) – 2D or 1D array. Dark-field image or a single row of it.
- **ratio** (*float*) – Ratio between exposure time used for recording projections and exposure time used for recording flat field.
- **use_dark** (*bool*) – Subtracting dark field if True. May no need in some cases.
- **options** (*dict, optional*) – Apply a zinger removal method and/or ring removal methods. E.g option1={"method": "dezinger", "para1": 0.001, "para2": 1}, option2={"method": "remove_stripe_based_sorting",

  "para1": 15, "para2": 1}

Returns *array_like* – 3D or 2D array. Corrected projections or corrected sinograms.

algotom.prep.correction.**generate_tilted_profile_chunk**(*mat*, *start_index*, *stop_index*, *angle*)
Generate a chunk of tilted horizontal intensity-profiles of an image.

Parameters

- **mat** (*array_like*) – 2D array.
- **start_index** (*int*) – Starting index of lines.
- **stop_index** (*int*) – Stopping index of lines.

- **angle** (*float*) – Tilted angle in degree.

Returns *array_like* – 2D array.

algotom.prep.correction.**generate_tilted_profile_line**(*mat*, *index*, *angle*)
Generate a tilted horizontal intensity-profile of an image.

Parameters

- **mat** (*array_like*) – 2D array.
- **index** (*int*) – Index of the line.
- **angle** (*float*) – Tilted angle in degree.

Returns *array_like* – 1D array.

algotom.prep.correction.**generate_tilted_sinogram**(*data*, *index*, *angle*, ***option*)
Generate a tilted sinogram of a 3D tomographic dataset or a hdf/nxs object.

Parameters

- **data** (*array_like or hdf object*) – 3D array.
- **index** (*int*) – Index of the sinogram.
- **angle** (*float*) – Tilted angle in degree.
- **option** (*list or tuple of int*) – To extract subset data along axis 0 from a hdf object. E.g option = (start, stop, step)

Returns *array_like* – 2D array. Tilted sinogram.

algotom.prep.correction.**generate_tilted_sinogram_chunk**(*data*, *start_index*, *stop_index*, *angle*, ***option*)
Generate a chunk of tilted sinograms of a 3D tomographic dataset or a hdf/nxs object.

Parameters

- **data** (*array_like or hdf object*) – 3D array.
- **start_index** (*int*) – Starting index of sinograms.
- **stop_index** (*int*) – Stopping index of sinograms.
- **angle** (*float*) – Tilted angle in degree.
- **option** (*list or tuple of int*) – To extract subset data along axis 0 from a hdf object. E.g option = (start, stop, step)

Returns *array_like* – 3D array. Chunk of tilted sinograms.

algotom.prep.correction.**mtf_deconvolution**(*mat*, *window*, *pad*)
Deconvolve an projection image using division in the Fourier domain. Window can be determined using the approach in Ref. [1].

Parameters

- **mat** (*array_like*) – 2D array. Projection image.
- **window** (*array_like*) – 2D array. MTF function.
- **pad** (*int*) – Padding width to reduce the side effects of the Fourier transform.

Returns *array_like* – 2D array. Deconvolved image.

References

algotom.prep.correction.**non_linear_function**(*intensity*, *q*, *n*, *opt=True*)
Function used to define the response curve.

Parameters

- **intensity** (*float*) – Values stay in the range of [0; 1]
- **q** (*float*) – Positive number.
- **n** (*float*) – Positive number. Must larger than 1.
- **opt** (*bool*) – True: Curve more to values closer to 1.0. False: Curve more to values closer to 0.0

Returns *float*

algotom.prep.correction.**unwarp_projection**(*proj*, *xcenter*, *ycenter*, *list_fact*)
Apply distortion correction to a projection image using the polynomial backward model (Ref. [1]).

Parameters

- **proj** (*array_like*) – 2D array. Projection image.
- **xcenter** (*float*) – Center of distortion in x-direction.
- **ycenter** (*float*) – Center of distortion in y-direction.
- **list_fact** (*list of float*) – Polynomial coefficients of the backward model.

Returns *array_like* – 2D array. Distortion corrected.

References

algotom.prep.correction.**unwarp_sinogram**(*data*, *index*, *xcenter*, *ycenter*, *list_fact*, ***option*)
Unwarp sinogram [:,index.:] of a 3D tomographic dataset or a hdf/nxs object.

Parameters

- **data** (*array_like or hdf object*) – 3D array.
- **index** (*int*) – Index of the sinogram.
- **xcenter** (*float*) – Center of distortion in x-direction.
- **ycenter** (*float*) – Center of distortion in y-direction.
- **list_fact** (*list of float*) – Polynomial coefficients of the backward model.
- **option** (*list or tuple of int*) – To extract subset data along axis 0 from a hdf object. E.g option = (start, stop, step)

Returns *array_like* – 2D array. Distortion-corrected sinogram.

algotom.prep.correction.**unwarp_sinogram_chunk**(*data*, *start_index*, *stop_index*, *xcenter*, *ycenter*, *list_fact*, ***option*)
Unwarp chunk of sinograms [:, start_index: stop_index, :] of a 3D tomographic dataset or a hdf/nxs object.

Parameters

- **data** (*array_like or hdf object*) – 3D array.
- **start_index** (*int*) – Starting index of sinograms.
- **stop_index** (*int*) – Stopping index of sinograms.

- **xcenter** (*float*) – Center of distortion in x-direction.
- **ycenter** (*float*) – Center of distortion in y-direction.
- **list_fact** (*list of float*) – Polynomial coefficients of the backward model.
- **option** (*list or tuple of int*) – To extract subset data along axis 0 from a hdf object. E.g option = [start, stop, step]

Returns *array_like* – 3D array. Distortion corrected.

1.5.7 algotom.prep.filtering

Module of filtering methods in the preprocessing stage: - Fresnel filter (denoising or low-pass filter). - Double-wedge filter.

Functions:

—

algotom.prep.filtering.**double_wedge_filter**(*sinogram*, *center=0*, *sino_type='180'*, *iteration=5*, *mask=None*, *ratio=1.0*, *pad=250*)

Apply double-wedge filter to a sinogram image (Ref. [1]).

Parameters

- **sinogram** (*array_like*) – 2D array. 180-degree sinogram or 360-degree sinogram.
- **center** (*float, optional*) – Center-of-rotation. No need for a 360-sinogram.
- **sino_type** (*{"180", "360"}*) – Sinogram type : 180-degree or 360-degree.
- **iteration** (*int*) – Number of iteration.
- **mask** (*array_like, optional*) – Double-wedge binary mask.
- **ratio** (*float, optional*) – Define the cut-off angle of the double-wedge filter.
- **pad** (*int*) – Padding width.

Returns *array_like* – 2D array. Filtered sinogram.

References

algotom.prep.filtering.**fresnel_filter**(*mat*, *ratio*, *dim=1*, *window=None*, *pad=150*, *apply_log=True*)

Apply a low-pass filter based on the Fresnel propagator to an image (Ref. [1]).

Parameters

- **mat** (*array_like*) – 2D array. Projection image or sinogram image.
- **ratio** (*float*) – Define the shape of the window. Larger is more smoothing.
- **dim** (*{1, 2}*) – Use "1" if working on a sinogram image and "2" if working on a projection image.
- **window** (*array_like, optional*) – Window for deconvolution.
- **pad** (*int*) – Padding width.
- **apply_log** (*bool, optional*) – Apply the logarithm function to the sinogram before filtering.

Returns *array_like* – 2D array. Filtered image.

References

algotom.prep.filtering.**make_double_wedge_mask**(*height*, *width*, *radius*)
Generate a double-wedge binary mask using Eq. (3) in Ref. [1]. Values outside the double-wedge region correspond to 0.0.

Parameters

- **height** (*int*) – Image height.
- **width** (*int*) – Image width.
- **radius** (*int*) – Radius of an object, in pixel unit.

Returns *array_like* – 2D binary mask.

References

algotom.prep.filtering.**make_fresnel_window**(*height*, *width*, *ratio*, *dim*)
Create a low pass window based on the Fresnel propagator. It is used to denoise a projection image (dim=2) or a sinogram image (dim=1).

Parameters

- **height** (*int*) – Image height
- **width** (*int*) – Image width
- **ratio** (*float*) – To define the shape of the window.
- **dim** (*{1, 2}*) – Use "1" if working on a sinogram image and "2" if working on a projection image.

Returns *array_like* – 2D array.

1.5.8 algotom.prep.removal

Module of removal methods in the preprocessing stage: - Many methods for removing stripe artifact in a sinogram (<-> ring artifact in a reconstructed image). - A zinger removal method. - Blob removal methods.

Functions:

—

algotom.prep.removal.**check_zinger_size**(*mat*, *max_size*)
Check if the size of a zinger is smaller than a given size.

Parameters

- **mat** (*array_like*) – 2D array.
- **max_size** (*int*) – Maximum size.

Returns *bool*

algotom.prep.removal.**generate_blob_mask**(*flat*, *size*, *snr*)
Generate a binary mask of blobs from a flat-field image (Ref. [1]).

Parameters

- **flat** (*array_like*) – 2D array. Flat-field image.
- **size** (*float*) – Estimated size of the largest blob.
- **snr** (*float*) – Ratio used to segment blobs.

Returns *array_like* – 2D array. Binary mask.

References

algotom.prep.removal.**remove_all_stripe**(*sinogram*, *snr=3.0*, *la_size=51*, *sm_size=21*, *drop_ratio=0.1*, *dim=1*, ***options*)

Remove all types of stripe artifacts in a sinogram by combining algorithm 6, 5, 4, and 3 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **snr** (*float*) – Ratio (>1.0) used to detect stripe locations. Greater is less sensitive.
- **la_size** (*int*) – Window size of the median filter to remove large stripes.
- **sm_size** (*int*) – Window size of the median filter to remove small-to-medium stripes.
- **drop_ratio** (*float, optional*) – Ratio of pixels to be dropped, which is used to to reduce the possibility of the false detection of stripes.
- **dim** (*{1, 2}, optional*) – Dimension of the window.
- **options** (*dict, optional*) – Use another smoothing filter rather than the median filter. E.g. options={"method": "gaussian_filter", "para1": (1,21))}

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_blob**(*mat*, *mask*)

Remove blobs in an image.

Parameters

- **mat** (*array_like*) – 2D array. Projection image or sinogram image.
- **mask** (*array_like*) – 2D binary mask.

Returns *array_like* – 2D array.

algotom.prep.removal.**remove_blob_1d**(*sino_1d*, *mask_1d*)

Remove blobs in one row of a sinogram, e.g. for a helical scan as shown in Ref. [1].

Parameters

- **sino_1d** (*array_like*) – 1D array. A row of a sinogram.
- **mask_1d** (*array_like*) – 1D binary mask.

Returns *array_like* – 1D array.

Notes

The method is used to remove streak artifacts caused by blobs in a sinogram generated from a helical-scan data [1].

References

algotom.prep.removal.**remove_dead_stripe**(*sinogram*, *snr=3.0*, *size=51*, *residual=True*)

Remove unresponsive or fluctuating stripe artifacts in a sinogram, algorithm 6 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **snr** (*float*) – Ratio (>1.0) used to detect stripe locations. Greater is less sensitive.
- **size** (*int*) – Window size of the median filter.
- **residual** (*bool, optional*) – Removing residual stripes if True.

Returns *ndarray* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_large_stripe**(*sinogram*, *snr=3.0*, *size=51*, *drop_ratio=0.1*, *norm=True*, ***options*)

Remove large stripe artifacts in a sinogram, algorithm 5 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image
- **snr** (*float*) – Ratio (>1.0) used to detect stripe locations. Greater is less sensitive.
- **size** (*int*) – Window size of the median filter.
- **drop_ratio** (*float, optional*) – Ratio of pixels to be dropped, which is used to to reduce the possibility of the false detection of stripes.
- **norm** (*bool, optional*) – Apply normalization if True.
- **options** (*dict, optional*) – Use another smoothing filter rather than the median filter. E.g. options={"method": "gaussian_filter", "para1": (1,21))}.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_2d_filtering_sorting**(*sinogram*, *sigma=3*, *size=21*, *dim=1*, ***options*)

Remove stripes using a 2D low-pass filter and the sorting-based technique, algorithm in section 3.3.4 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **sigma** (*int*) – Sigma of the Gaussian window.
- **size** (*int*) – Window size of the median filter.

- **dim** (*{1, 2}, optional*) – Dimension of the window.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_fft**(*sinogram*, *u=20*, *n=8*, *v=1*, *sort=False*)

Remove stripes using the method in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **u** (*int*) – Cutoff frequency.
- **n** (*int*) – Filter order.
- **v** (*int*) – Number of rows (* 2) to be applied the filter.
- **sort** (*bool, optional*) – Apply sorting (Ref. [2]) if True.

Returns *ndarray* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_filtering**(*sinogram*, *sigma=3*, *size=21*, *dim=1*, *sort=True*, ***options*)

Remove stripe artifacts in a sinogram using the filtering technique, algorithm 2 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image
- **sigma** (*int*) – Sigma of the Gaussian window used to separate the low-pass and high-pass components of the intensity profile of each column.
- **size** (*int*) – Window size of the median filter.
- **dim** (*{1, 2}, optional*) – Dimension of the window.
- **sort** (*bool, optional*) – Apply sorting if True.
- **options** (*dict, optional*) – Use another smoothing filter rather than the median filter. E.g. options={"method": "gaussian_filter", "para1": (1,21))}.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_fitting**(*sinogram*, *order=2*, *sigma=10*, *sort=False*, *num_chunk=1*, ***options*)

Remove stripe artifacts in a sinogram using the fitting technique, algorithm 1 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image
- **order** (*int*) – Polynomial fit order.
- **sigma** (*int*) – Sigma of the Gaussian window in the x-direction. Smaller is stronger.

- **sort** (*bool, optional*) – Apply sorting if True.
- **num_chunk** (*int*) – Number of chunks of rows to apply the fitting.
- **options** (*dict, optional*) – Use another smoothing filter rather than the Fourier gaussian filter. E.g. options={"method": "gaussian_filter", "para1": (1,21))}.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_interpolation**(*sinogram*, *snr=3.0*, *size=51*, *drop_ratio=0.1*, *norm=True*, *kind='linear'*, ***options*)

Combination of algorithm 4, 5, and 6 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image
- **snr** (*float*) – Ratio (>1.0) used to detect stripe locations. Greater is less sensitive.
- **size** (*int*) – Window size of the median filter used to detect stripes.
- **drop_ratio** (*float, optional*) – Ratio of pixels to be dropped, which is used to to reduce the possibility of the false detection of stripes.
- **norm** (*bool, optional*) – Apply normalization if True.
- **kind** (*{'linear', 'cubic', 'quintic'}, optional*) – The kind of spline interpolation to use. Default is 'linear'.
- **options** (*dict, optional*) – Use another smoothing filter rather than the median filter. E.g. options={"method": "gaussian_filter", "para1": (1,21))}

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_normalization**(*sinogram*, *sigma=15*, *num_chunk=1*, *sort=True*, ***options*)

Remove stripes using the method in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **sigma** (*int*) – Sigma of the Gaussian window.
- **num_chunk** (*int*) – Number of chunks of rows.
- **sort** (*bool, optional*) – Apply sorting (Ref. [2]) if True.
- **options** (*dict, optional*) – Use another smoothing 1D-filter rather than the Gaussian filter. E.g. options={"method": "median_filter", "para1": 21)}.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_regularization**(*sinogram*, *alpha=0.0005*, *num_chunk=1*, *apply_log=True*, *sort=True*)

Remove stripes using the method in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **alpha** (*float*) – Regularization parameter, e.g. 0.0005. Smaller is stronger.
- **num_chunk** (*int*) – Number of chunks of rows.
- **apply_log** (*bool*) – Apply the logarithm function to the sinogram if True.
- **sort** (*bool, optional*) – Apply sorting (Ref. [2]) if True.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_sorting**(*sinogram*, *size=21*, *dim=1*, ***options*)

Remove stripe artifacts in a sinogram using the sorting technique, algorithm 3 in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **size** (*int*) – Window size of the median filter.
- **dim** (*{1, 2}, optional*) – Dimension of the window.
- **options** (*dict, optional*) – Use another smoothing filter rather than the median filter. E.g. options={"method": "gaussian_filter", "para1": (1,21))}

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_stripe_based_wavelet_fft**(*sinogram*, *level=5*, *size=1*, *wavelet_name='db9'*, *window_name='gaussian'*, *sort=False*, ***options*)

Remove stripes using the method in Ref. [1]. Angular direction is along the axis 0.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **level** (*int*) – Wavelet decomposition level.
- **size** (*int*) – Damping parameter. Larger is stronger.
- **wavelet_name** (*str*) – Name of a wavelet. Search pywavelets API for a full list.
- **window_name** (*str*) – High-pass window. Two options: "gaussian" or "butter".
- **sort** (*bool, optional*) – Apply sorting (Ref. [2]) if True.

Returns *array_like* – 2D array. Stripe-removed sinogram.

References

algotom.prep.removal.**remove_zinger**(*mat*, *threshold*, *size=2*)
Remove zinger using the method in Ref. [1], working on a projection image or sinogram image.

Parameters

- **mat** (*array_like*) – 2D array. Projection image or sinogram image.
- **threshold** (*float*) – Threshold to segment zingers. Smaller is more sensitive. Recommended range [0.05, 0.1].
- **size** (*int*) – Size of a zinger.

Returns *array_like* – 2D array. Zinger-removed image.

References

algotom.prep.removal.**select_zinger**(*mat*, *max_size*)
Select zingers smaller than a certain size.

Parameters

- **mat** (*array_like*) – 2D array.
- **max_size** (*int*) – Maximum size in pixel.

Returns *array_like* – 2D binary array.

1.5.9 algotom.rec.reconstruction

Module of FFT-based reconstruction methods in the reconstruction stage: - Filtered back-projection (FBP) method for GPU (using numba and cuda) and CPU. - Direct Fourier inversion (DFI) method. - Wrapper for Astra Toolbox reconstruction (optional) - Wrapper for Tomopy-gridrec reconstruction (optional)

Functions:

—

algotom.rec.reconstruction.**apply_ramp_filter**(*sinogram*, *ramp_win=None*, *filter_name=None*, *pad=None*, *pad_mode='edge'*)
Apply the ramp filter to a sinogram with the option of adding a smoothing filter.

Parameters

- **sinogram** (*array_like*) – 2D rray. Sinogram image.
- **ramp_win** (*complex ndarray or None*) – Ramp window in the Fourier space.
- **filter_name** (*{None, “hann”, “bartlett”, “blackman”, “hamming”, “nuttall”,*) – “parzen”, “triang”} Name of a smoothing window used.
- **pad** (*int or None*) – To apply padding before the FFT. The value is set to 10% of the image width if None is given.
- **pad_mode** (*str*) – Padding method. Full list can be found at numpy.pad documentation.

Returns *array_like* – Filtered sinogram.

algotom.rec.reconstruction.**astra_reconstruction**(*sinogram*, *center*, *angles=None*, *ratio=1.0*, *method='FBP_CUDA'*, *num_iter=1*, *filter_name='hann'*, *pad=None*, *apply_log=True*)

Wrapper of reconstruction methods implemented in the astra toolbox package. https://www.astra-toolbox.com/docs/algs/index.html Users must install Astra Toolbox before using this function.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **center** (*float*) – Center of rotation.
- **angles** (*array_like*) – 1D array. List of angles (radian) corresponding to the sinogram.
- **ratio** (*float*) – To apply a circle mask to the reconstructed image.
- **method** (*str*) – Reconstruction algorithms. for CPU: 'FBP', 'SIRT', 'SART', 'ART', 'CGLS'. for GPU: 'FBP_CUDA', 'SIRT_CUDA', 'SART_CUDA', 'CGLS_CUDA'.
- **num_iter** (*int*) – Number of iterations if using iteration methods.
- **filter_name** (*str*) – Apply filter if using FBP method. Options: 'hamming', 'hann', 'lanczos', 'kaiser', 'parzen',...
- **pad** (*int*) – Padding to reduce the side effect of FFT.
- **apply_log** (*bool*) – Apply the logarithm function to the sinogram before reconstruction.

Returns *array_like* – Square array.

algotom.rec.reconstruction.**back_projection_cpu**(*sinogram*, *angles*, *xlist*, *center*)

Implement the back-projection algorithm using CPU.

sinogram [array_like] 2D array. (Filtered) sinogram image.

angles [array_like] 1D array. Angles (radian) corresponding to the sinogram.

xlist [array_like] 1D array. Distances of the integration lines to the image center.

center [float] Center of rotation.

Returns **recon** (*array_like*) – Square array. Reconstructed image.

algotom.rec.reconstruction.**dfi_reconstruction**(*sinogram*, *center*, *angles=None*, *ratio=1.0*, *filter_name='hann'*, *pad_rate=0.25*, *pad_mode='edge'*, *apply_log=True*)

Apply the DFI (direct Fourier inversion) reconstruction method to a sinogram image (Ref. [1]). The method is a practical and direct implementation of the Fourier slice theorem (Ref. [2]).

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **center** (*float*) – Center of rotation.
- **angles** (*array_like*) – 1D array. List of angles (in radian) corresponding to the sinogram.
- **ratio** (*float*) – To apply a circle mask to the reconstructed image.
- **filter_name** (*{None, "hann", "bartlett", "blackman", "hamming", "nuttall",*) – "parzen", "triang"} Apply a smoothing filter.
- **pad_rate** (*float*) – To apply padding before the FFT. The padding width equals to (pad_rate * image_width).
- **pad_mode** (*str*) – Padding method. Full list can be found at numpy.pad documentation.

- **apply_log** (*bool*) – Apply the logarithm function to the sinogram before reconstruction.

Returns *array_like* – Square array. Reconstructed image.

References

algotom.rec.reconstruction.**fbp_reconstruction**(*sinogram*, *center*, *angles=None*, *ratio=1.0*, *ramp_win=None*, *filter_name='hann'*, *pad=None*, *pad_mode='edge'*, *apply_log=True*, *gpu=True*)

Apply the FBP (filtered back-projection) reconstruction method to a sinogram image.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **center** (*float*) – Center of rotation.
- **angles** (*array_like, optional*) – 1D array. List of angles (in radian) corresponding to the sinogram.
- **ratio** (*float, optional*) – To apply a circle mask to the reconstructed image.
- **ramp_win** (*complex ndarray, optional*) – Ramp window in the Fourier space. It will be generated if None is given.
- **filter_name** (*{None, "hann", "bartlett", "blackman", "hamming", "nuttall",*) – "parzen", "triang"} Apply a smoothing filter.
- **pad** (*int, optional*) – To apply padding before the FFT. The value is set to 10% of the image width if None is given.
- **pad_mode** (*str, optional*) – Padding method. Full list can be found at numpy.pad documentation.
- **apply_log** (*bool, optional*) – Apply the logarithm function to the sinogram before reconstruction.
- **gpu** (*bool, optional*) – Use GPU for computing if True.

Returns *array_like* – Square array. Reconstructed image.

algotom.rec.reconstruction.**generate_mapping_coordinate**(*width_sino*, *height_sino*, *width_rec*, *height_rec*)

Calculate coordinates in the sinogram space from coordinates in the reconstruction space (in the Fourier domain). They are used for the DFI (direct Fourier inversion) reconstruction method.

Parameters

- **width_sino** (*int*) – Width of a sinogram image.
- **height_sino** (*int*) – Height of a sinogram image.
- **width_rec** (*int*) – Width of a reconstruction image.
- **height_rec** (*int*) – Height of a reconstruction image.

Returns

- **r_mat** (*array_like*) – 2D array. Broadcast of the r-coordinates.
- **theta_mat** (*array_like*) – 2D array. Broadcast of the theta-coordinates.

algotom.rec.reconstruction.**gridrec_reconstruction**(*sinogram*, *center*, *angles=None*, *ratio=1.0*, *filter_name='shepp'*, *apply_log=True*, *pad=True*, *ncore=1*)

Wrapper of the gridrec method implemented in the tomopy package: https://tomopy.readthedocs.io/en/latest/api/tomopy.recon.algorithm.html Users must install Tomopy before using this function.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **center** (*float*) – Center of rotation.
- **angles** (*array_like*) – 1D array. List of angles (radian) corresponding to the sinogram.
- **ratio** (*float*) – To apply a circle mask to the reconstructed image.
- **filter_name** (*str*) – Apply a smoothing filter. Full list is at: https://github.com/tomopy/tomopy/blob/master/source/tomopy/recon/algorithm.py
- **apply_log** (*bool*) – Apply the logarithm function to the sinogram before reconstruction.
- **pad** (*bool*) – Apply edge padding to the nearest power of 2.

Returns *array_like* – Square array.

algotom.rec.reconstruction.**make_2d_ramp_window**(*height*, *width*, *filter_name=None*)

Make the 2d ramp window (in the Fourier space) by repeating the 1d ramp window with the option of adding a smoothing window.

Parameters

- **height** (*int*) – Height of the window.
- **width** (*int*) – Width of the window.
- **filter_name** (*{None, “hann”, “bartlett”, “blackman”, “hamming”, “nuttall”,*) – “parzen”, “triang”} Name of a smoothing window used.

Returns *complex ndarray* – 2D array.

algotom.rec.reconstruction.**make_smoothing_window**(*filter_name*, *width*)

Make a 1d smoothing window.

Parameters

- **filter_name** (*{“hann”, “bartlett”, “blackman”, “hamming”, “nuttall”,*) – “parzen”, “triang”} Window function used for filtering.
- **width** (*int*) – Width of the window.

Returns *array_like* – 1D array.

1.5.10 algotom.util.calibration

Module of calibration methods: - Correcting the non-uniform background of an image. - Binarizing an image. - Calculating the distance between two point-like objects segmented from

two images. Useful for determining pixel-size in helical scans.

Functions:

—

algotom.util.calibration.**binarize_image**(*mat*, *threshold=None*, *bgr='bright'*, *norm=False*, *denoise=True*, *invert=True*)

Binarize an image.

Parameters

- **mat** (*array_like*) – 2D array.
- **threshold** (*float, optional*) – Threshold value for binarization. Automatically calculated using Algorithm 4 in Ref. [1] if None.
- **bgr** (*{"bright", "dark"}*) – To indicate the brightness of the background against image features.
- **norm** (*bool, optional*) – Apply normalization if True.
- **denoise** (*bool, optional*) – Apply denoising if True.
- **invert** (*bool, optional*) – Invert the contrast if needed.

Returns *array_like* – 2D binary array.

References

algotom.util.calibration.**calculate_distance**(*mat1*, *mat2*, *size_opt='max'*, *threshold=None*, *bgr='bright'*, *norm=False*, *denoise=True*, *invert=True*)

Calculate the distance between two point-like objects segmented from two images. Useful for measuring pixel-size in helical scans (Ref. [1]).

Parameters

- **mat1** (*array_like*) – 2D array.
- **mat2** (*array_like*) – 2D array.
- **size_opt** (*{"max", "min", "median", "mean"}*) – Options to select binary objects based on their size.
- **threshold** (*float, optional*) – Threshold value for binarization. Automatically calculated using Algorithm 4 in Ref. [2] if None.
- **bgr** (*{"bright", "dark"}*) – To indicate the brightness of the background against image features.
- **norm** (*bool, optional*) – Apply normalization if True.
- **denoise** (*bool, optional*) – Apply denoising if True.
- **invert** (*bool, optional*) – Invert the contrast if needed.

References

algotom.util.calibration.**calculate_threshold**(*mat*, *bgr='bright'*)
Calculate threshold value based on Algorithm 4 in Ref. [1].

Parameters

- **mat** (*array_like*) – 2D array.
- **bgr** (*{"bright", "dark"}*) – To indicate the brightness of the background against image features.

Returns *float* – Threshold value.

References

algotom.util.calibration.**check_dot_size**(*mat*, *min_size*, *max_size*)
Check if the size of a dot is in a range.

Parameters

- **mat** (*array_like*) – 2D array.
- **min_size** (*float*) – Minimum size.
- **max_size** (*float*) – Maximum size.

Returns *bool*

algotom.util.calibration.**get_dot_size**(*mat*, *size_opt='max'*)
Get size of binary dots given the option.

Parameters

- **mat** (*array_like*) – 2D binary array.
- **size_opt** (*{"max", "min", "median", "mean"}*) – Select options.

Returns **dot_size** (*float*) – Size of the dot.

algotom.util.calibration.**invert_dot_contrast**(*mat*)
Invert the contrast of a 2D binary array to make sure that a dot is white.

Parameters **mat** (*array_like*) – 2D binary array.

Returns *array_like* – 2D array.

algotom.util.calibration.**normalize_background**(*mat*, *radius=51*)
Correct a non-uniform background of an image using the median filter.

Parameters

- **mat** (*array_like*) – 2D array.
- **radius** (*int*) – Size of the median filter.

Returns *array_like* – 2D array. Corrected image.

algotom.util.calibration.**normalize_background_based_fft**(*mat*, *sigma=5*, *pad=None*, *mode='reflect'*)
Correct a non-uniform background of an image using a Fourier Gaussian filter.

Parameters

- **mat** (*array_like*) – 2D array.
- **sigma** (*int*) – Sigma of the Gaussian.

- **pad** (*int*) – Padding for the Fourier transform.
- **mode** (*str, list of str, or tuple of str*) – Padding method. One of options : 'reflect', 'edge', 'constant'. Full list is at: https://numpy.org/doc/stable/reference/generated/numpy.pad.html

Returns *array_like* – 2D array. Corrected image.

algotom.util.calibration.**select_dot_based_size**(*mat*, *dot_size*, *ratio=0.01*)
Select dots having a certain size.

Parameters

- **mat** (*array_like*) – 2D array.
- **dot_size** (*float*) – Size of the standard dot.
- **ratio** (*float*) – Used to calculate the acceptable range. [dot_size - ratio*dot_size; dot_size + ratio*dot_size]

Returns *array_like* – 2D array. Selected dots.

1.5.11 algotom.util.config

Functions:

—

class algotom.util.config.**Params**(*sections=()*)
Bases: object

add_arguments(*parser*)

add_parser_args(*parser*)

get_defaults()

algotom.util.config.**config_to_list**(*config_name='/home/docs/algotom.conf'*)
Read arguments from config file and convert them to a list of keys and values as sys.argv does when they are specified on the command line. *config_name* is the file name of the config file.

algotom.util.config.**get_config_name**()
Get the command line –config option.

algotom.util.config.**log_values**(*args*)
Log all values set in the args namespace.

Arguments are grouped according to their section and logged alphabetically using the DEBUG log level thus –verbose is required.

algotom.util.config.**parse_known_args**(*parser*, *subparser=False*)
Parse arguments from file and then override by the ones specified on the command line. Use *parser* for parsing and is *subparser* is True take into account that there is a value on the command line specifying the subparser.

algotom.util.config.**show_config**(*args*)
Log all values set in the args namespace.

Arguments are grouped according to their section and logged alphabetically using the DEBUG log level thus –verbose is required.

algotom.util.config.**write**(*config_file*, *args=None*, *sections=None*)
Write *config_file* with values from *args* if they are specified, otherwise use the defaults. If *sections* are specified, write values from *args* only to those sections, use the defaults on the remaining ones.

1.5.12 algotom.util.log

tomoscan custom logger

Functions:

—

class algotom.util.log.**ColoredLogFormatter**(*fmt*, *datefmt=None*, *style='%'*)
Bases: logging.Formatter

formatMessage(*record*)

algotom.util.log.**debug**(*msg*, **args*, ***kwargs*)

algotom.util.log.**error**(*msg*, **args*, ***kwargs*)

algotom.util.log.**info**(*msg*, **args*, ***kwargs*)

algotom.util.log.**setup_custom_logger**(*lfname=None*, *stream_to_console=True*)

algotom.util.log.**warning**(*msg*, **args*, ***kwargs*)

1.5.13 algotom.util.simulation

Module of simulation methods: 1- Methods for designing a customized 2D phantom. 2- Method for calculating a sinogram of a phantom based on the Fourier

slice theorem.

3- Methods for adding artifacts to a simulated sinogram.

Functions:

—

algotom.util.simulation.**add_background_fluctuation**(*sinogram*, *strength_ratio=0.2*)
Fluctuate the background of a sinogram image using a Gaussian profile beam.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **strength_ratio** (*float*) – To define the strength of the variation. The value is in the range of [0.0, 1.0].

Returns *array_like*

algotom.util.simulation.**add_noise**(*mat*, *noise_ratio=0.1*)
Add Gaussian noise to an image.

Parameters

- **mat** (*array_like*) – 2D array
- **noise_ratio** (*float*) – Ratio between the noise level and the mean of the array.

Returns *array_like*

algotom.util.simulation.**add_stripe_artifact**(*sinogram*, *size*, *position*, *strength_ratio=0.2*, *stripe_type='partial'*)

Add stripe artifacts to a sinogram.

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image.
- **size** (*int*) – Size of stripe artifact.
- **position** (*int*) – Position of the stripe.
- **strength_ratio** (*float*) – To define the strength of the artifact. The value is in the range of [0.0, 1.0].
- **stripe_type** (*{"partial", "full", "dead", "fluctuating"}*) – Type of stripe as classified in Ref. [1].

Returns *array_like*

References

algotom.util.simulation.**convert_to_Xray_image**(*sinogram*, *global_max=None*)

Convert a simulated sinogram to an equivalent X-ray image.

Parameters

- **sinogram** (*array_like*) – 2D array.
- **global_max** (*float*) – Maximum value used for normalizing array values to stay in the range of [0.0, 1.0].

Returns *array_like*

algotom.util.simulation.**make_elliptic_mask**(*size*, *center*, *length*, *angle*)

Create an elliptic mask.

Parameters

- **size** (*int*) – Size of a square array.
- **center** (*float or tuple of float*) – Ellipse center.
- **length** (*float or tuple of float*) – Lengths of ellipse axes.
- **angle** (*float*) – Rotation angle (Degree) of the ellipse.

Returns *array_like* – Square array.

algotom.util.simulation.**make_face_phantom**(*size*)

Create a face phantom for testing reconstruction methods.

Parameters **size** (*int*) – Size of a square array.

Returns *array_like* – Square array.

algotom.util.simulation.**make_line_target**(*size*)

Create line patterns for testing the resolution of a reconstructed image.

Parameters **size** (*int*) – Size of a square array.

Returns *array_like* – Square array.

algotom.util.simulation.**make_rectangular_mask**(*size*, *center*, *length*, *angle*)

Create a rectangular mask.

Parameters

- **size** (*int*) – Size of a square array.
- **center** (*float or tuple of float*) – Center of the mask.
- **length** (*float or tuple of float*) – Lengths of the rectangular mask.
- **angle** (*float*) – Rotation angle (Degree) of the mask.

Returns *array_like* – Square array.

algotom.util.simulation.**make_sinogram**(*mat*, *angles*, *pad_rate=0.5*, *pad_mode='edge'*)
Create a sinogram (series of 1D projections) from a 2D image based on the Fourier slice theorem (Ref. [1]).

Parameters

- **mat** (*array_like*) – Square array.
- **angles** (*array_like*) – 1D array. List of angles (in radian) for projecting.
- **pad_rate** (*float*) – To apply padding before the FFT. The padding width equals to (pad_rate * image_width).
- **pad_mode** (*str*) – Padding method. Full list can be found at numpy.pad documentation.

References

algotom.util.simulation.**make_triangular_mask**(*size*, *center*, *length*, *angle*)
Create an isosceles triangle mask.

Parameters

- **size** (*int*) – Size of a square array.
- **center** (*float or tuple of float*) – Center of the mask.
- **length** (*float or tuple of float*) – Lengths of the mask.
- **angle** (*float*) – Rotation angle (Degree) of the mask.

Returns *array_like* – Square array.

1.5.14 algotom.util.utility

Module of utility methods: 1- Methods for parallel computing, geometric transformation, masking. 2- Methods for customizing stripe/ring removal methods

2.1 - sort_forward 2.2 - sort_backward 2.3 - separate_frequency_component 2.4 - generate_fitted_image 2.5 - detect_stripe 2.6 - calculate_regularization_coefficient 2.7 - make_2d_butterworth_window 2.8 - make_2d_damping_window 2.9 - apply_wavelet_decomposition 2.10 - apply_wavelet_reconstruction 2.11 - apply_filter_to_wavelet_component 2.12 - interpolate_inside_stripe 2.13 - transform_slice_forward 2.14 - transform_slice_backward

3- Customized smoothing filters: 3.1 - apply_gaussian_filter (in the Fourier space) 3.2 - apply_regularization_filter

4- Methods for grid scans: 4.1 - detect_sample 4.2 - fix_non_sample_areas 4.3 - locate_slice 4.4 - locate_slice_chunk

Functions:

—

algotom.util.utility.**apply_1d_regularizer**(*list_data*, *sijmat*)
Supplementary method for the method of "apply_regularization_filter". To apply a regularizer to an 1D-array.

algotom.util.utility.**apply_filter_to_wavelet_component**(*data*, *level=None*, *order=1*, *method='gaussian_filter'*, *para=[(1, 11)]*)
Apply a filter to a component of the wavelet decomposition of an image.

Parameters

- **data** (*list or tuple*) – The first element is an 2D-array, next elements are tuples of three 2D-arrays. i.e [mat_n, (cH_level_n, cV_level_n, cD_level_n), …, (cH_level_1, cV_level_1, cD_level_1)].
- **level** (*int, list of int, or None*) – Decomposition level to be applied the filter.
- **order** (*{0, 1, 2}*) – Specify which component in a tuple, (cH_level_n, cV_level_n, cD_level_n) to be filtered.
- **method** (*str*) – Name of the filter in the namespace.
- **para** (*list or tuple*) – Parameters of the filter.

Returns *list or tuple* – The first element is an 2D-array, next elements are tuples of three 2D-arrays. i.e [mat_n, (cH_level_n, cV_level_n, cD_level_n), …, (cH_level_1, cV_level_1, cD_level_1)].

algotom.util.utility.**apply_gaussian_filter**(*mat*, *sigmax*, *sigmay*, *pad=None*, *mode=None*)
Filtering an image in the Fourier domain using a 2D Gaussian window. Smaller is stronger.

Parameters

- **mat** (*array_like*) – 2D array.
- **sigmax** (*int*) – Sigma in the x-direction.
- **sigmay** (*int*) – Sigma in the y-direction.
- **pad** (*int or None*) – Padding for the Fourier transform.
- **mode** (*str, list of str, or tuple of str*) – Padding method. One of options : 'reflect', 'edge', 'constant'. Full list is at: https://numpy.org/doc/stable/reference/generated/numpy.pad.html

Returns *array_like* – 2D array. Filtered image.

algotom.util.utility.**apply_method_to_multiple_sinograms**(*data*, *method*, *para*, *ncore=None*)
Apply a processing method (in "filtering", "removal", and "reconstruction" module) to multiple sinograms or multiple slices in parallel.

Parameters

- **data** (*array_like or hdf object*) – 3D array data where sinograms/slices are extracted along axis 1, e.g [:, i, :].
- **method** (*str*) – Name of a method. e.g. "remove_stripe_based_sorting".
- **para** (*list*) – Parameters of the method. e.g. [21, 1]
- **ncore** (*int or None*) – Number of cores used for computing. Automatically selected if None.

Returns *array_like* – Same axis-definition as the input.

algotom.util.utility.**apply_regularization_filter**(*mat*, *alpha*, *axis=1*, *ncore=None*)
Apply a regularization filter using the method in Ref. [1]. Note that it's computationally costly.

Parameters

- **mat** (*array_like*) – 2D array
- **alpha** (*float*) – Regularization parameter, e.g. 0.001. Smaller is stronger.
- **axis** (*int*) – Axis along which to apply the filter.
- **ncore** (*int or None*) – Number of cores used for computing. Automatically selected if None.

Returns *array_like* – 2D array. Smoothed image.

References

algotom.util.utility.**apply_wavelet_decomposition**(*mat*, *wavelet_name*, *level=None*)
Apply 2D wavelet decomposition.

Parameters

- **mat** (*array_like*) – 2D array.
- **wavelet_name** (*str*) – Name of a wavelet. E.g. "db5"
- **level** (*int, optional*) – Decomposition level. It is constrained to return an array with a minimum size of larger than 16 pixels.

Returns *list* – The first element is an 2D-array, next elements are tuples of three 2D-arrays. i.e [mat_n, (cH_level_n, cV_level_n, cD_level_n), ..., (cH_level_1, cV_level_1, cD_level_1)]

algotom.util.utility.**apply_wavelet_reconstruction**(*data*, *wavelet_name*, *ignore_level=None*)
Apply 2D wavelet reconstruction.

Parameters

- **data** (*list or tuple*) – The first element is an 2D-array, next elements are tuples of three 2D-arrays. i.e [mat_n, (cH_level_n, cV_level_n, cD_level_n), ..., (cH_level_1, cV_level_1, cD_level_1)].
- **wavelet_name** (*str*) – Name of a wavelet. E.g. "db5"
- **ignore_level** (*int, optional*) – Decomposition level to be ignored for reconstruction.

Returns *array_like* – 2D array. Note that the sizes of the array are always even numbers.

algotom.util.utility.**calculate_regularization_coefficient**(*width*, *alpha*)
Calculate coefficients used for the regularization-based method. Eq. (7) in Ref. [1].

Parameters

- **width** (*int*) – Width of a square array.
- **alpha** (*float*) – Regularization parameter.

Returns *float* – Square array.

References

algotom.util.utility.**check_level**(*level*, *n_level*)
Supplementary method for the method of "apply_filter_to_wavelet_component". To check if the provided level is in the right format.

algotom.util.utility.**detect_sample**(*sinogram*, *sino_type='180'*)
To check if there is a sample in a sinogram using the "double-wedge" property of the Fourier transform of the sinogram (Ref. [1]).

Parameters

- **sinogram** (*array_like*) – 2D array. Sinogram image
- **sino_type** (*{"180", "360"}*) – Sinogram type : 180-degree or 360-degree.

Returns *bool* – True if there is a sample.

References

algotom.util.utility.**detect_stripe**(*list_data*, *snr*)
Locate stripe positions using Algorithm 4 in Ref. [1]

Parameters

- **list_data** (*array_like*) – 1D array. Normalized data.
- **snr** (*float*) – Ratio (>1.0) used to detect stripe locations. Greater is less sensitive.

Returns *array_like* – 1D binary mask.

References

algotom.util.utility.**fix_non_sample_areas**(*overlap_metadata*)
Used to fix overlap values of grid-cells without sample by copying from its neighbours

Parameters **overlap_metadata** (*array_like*) – A matrix of overlap values of each grid-cell where each element is a list of [overlap, side].

Returns **metadata** (*array_like*)

algotom.util.utility.**generate_fitted_image**(*mat*, *order*, *axis=0*, *num_chunk=1*)
Apply a polynomial fitting along an axis of an image. e.g. axis=0 is to apply the fitting to each column.

Parameters

- **mat** (*array_like*) – 2D array.
- **order** (*int*) – Order of the polynomial used to fit.
- **axis** (*int*) – Axis along which to apply the filter.
- **num_chunk** (*int*) – Number of chunks of rows or columns to apply the fitting.

Returns **mat_fit** (*array_like*)

algotom.util.utility.**interpolate_inside_stripe**(*mat*, *list_mask*, *kind='linear'*)
Interpolate gray-scales inside vertical stripes of an image. Stripe locations given by a binary 1D-mask.

Parameters

- **mat** (*array_like*) – 2D array.

- **list_mask** (*array_like*) – 1D array. Must equal the width of an image.
- **kind** (*{'linear', 'cubic', 'quintic'}, optional*) – The kind of spline interpolation to use. Default is 'linear'.

Returns *array_like*

algotom.util.utility.**locate_slice**(*slice_idx*, *height*, *overlap_metadata*)
Locate slice indices in grid-rows given a slice index of the reconstruction data as a whole.

Parameters

- **slice_idx** (*int*) – Slice index of full reconstruction data.
- **height** (*int*) – Height of a projection image of each grid-cell.
- **overlap_metadata** (*array_like*) – A matrix of overlap values of each grid-row where each element is a list of [overlap, side]. Used to stitch the grid-data along the row-direction.

Returns *list of int and float* – If the slice is not in the overlapping area between two grid-rows, the result is a list of [grid_row_index, slice_index, weight_factor]. If the slice is in the overlapping area between two grid-rows, the result is a list of [[grid_row_index_0, slice_index_0, weight_factor_0], [grid_row_index_1, slice_index_1, weight_factor_1]]

algotom.util.utility.**locate_slice_chunk**(*slice_start*, *slice_stop*, *height*, *overlap_metadata*)
Locate slice indices in grid-rows given slice indices of the reconstruction data as a whole.

Parameters

- **slice_start** (*int*) – Starting index of full reconstruction data.
- **slice_stop** (*int*) – Stopping index of full reconstruction data.
- **height** (*int*) – Height of a projection image of each grid-cell.
- **overlap_metadata** (*array_like*) – A matrix of overlap values of each grid-row where each element is a list of [overlap, side]. Used to stitch the grid-data along the row-direction.

Returns *list of list of int and float* – List of results for each slice index. If a slice is not in the overlapping area between two grid-rows, the result is a list of [grid_row_index, slice_index, weight_factor]. If a slice is in the overlapping area between two grid-rows, the result is a list of [[grid_row_index_0, slice_index_0, weight_factor_0], [grid_row_index_1, slice_index_1, weight_factor_1]].

algotom.util.utility.**make_2d_butterworth_window**(*width*, *height*, *u*, *v*, *n*)
Create a 2d window from the 1D Butterworth window.

Parameters

- **height** (*int*) – Height of the window.
- **width** (*int*) – Width of the window.
- **u** (*int*) – Cutoff frequency.
- **n** (*int*) – Filter order.
- **v** (*int*) – Number of rows (= 2*v) around the height middle are the 1D Butterworth windows.

Returns *array_like* – 2D array.

algotom.util.utility.**make_2d_damping_window**(*width*, *height*, *size*, *window_name='gaussian'*)
Make 2D damping window from a list of 1D window for a Fourier-space filter, i.e. a high-pass filter.

Parameters

- **height** (*int*) – Height of the window.

- **width** (*int*) – Width of the window.
- **size** (*int*) – Sigma of a Gaussian window or cutoff frequency of a Butterworth window.
- **window_name** (*str, optional*) – Two options: "gaussian" or "butter".

Returns *array_like* – 2D array of the window.

algotom.util.utility.**make_2d_gaussian_window**(*height*, *width*, *sigmax*, *sigmay*)
Create a 2D Gaussian window.

Parameters

- **height** (*int*) – Height of the image.
- **width** (*int*) – Width of the image.
- **sigmax** (*int*) – Sigma in the x-direction.
- **sigmay** (*int*) – Sigma in the y-direction.

Returns *array_like* – 2D array.

algotom.util.utility.**make_circle_mask**(*width*, *ratio*)
Create a circle mask.

Parameters

- **width** (*int*) – Width of a square array.
- **ratio** (*float*) – Ratio between the diameter of the mask and the width of the array.

Returns *array_like* – Square array.

algotom.util.utility.**mapping**(*mat*, *xmat*, *ymat*, *order=1*, *mode='reflect'*)
Apply a geometric transformation to a 2D array

Parameters

- **mat** (*array_like*) – 2D array.
- **xmat** (*array_like*) – 2D array of the x-coordinates.
- **ymat** (*array_like*) – 2D array of the y-coordinates.
- **order** (*int, optional*) – The order of the spline interpolation, default is 1. The order has to be in the range 0-5.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The mode parameter determines how the input array is extended beyond its boundaries. Default is 'reflect'.

Returns *array_like* – 2D array.

algotom.util.utility.**polar_from_rectangular**(*width_pol*, *height_pol*, *width_reg*, *height_reg*)
Generate polar coordinates from grid coordinates.

Parameters

- **width_pol** (*int*) – Width of an image in the polar coordinate system.
- **height_pol** (*int*) – Height of an image in the polar coordinate system.
- **width_reg** (*int*) – Width of an image in the Cartesian coordinate system.
- **height_reg** (*int*) – Height of an image in the Cartesian coordinate system.

Returns

- **r_mat** (*array_like*) – 2D array. Broadcast of the r-coordinates.

- **theta_mat** (*array_like*) – 2D array. Broadcast of the theta-coordinates.

algotom.util.utility.**rectangular_from_polar**(*width_reg*, *height_reg*, *width_pol*, *height_pol*)
Generate coordinates of a rectangular grid from polar coordinates.

Parameters

- **width_reg** (*int*) – Width of an image in the Cartesian coordinate system.
- **height_reg** (*int*) – Height of an image in the Cartesian coordinate system.
- **width_pol** (*int*) – Width of an image in the polar coordinate system.
- **height_pol** (*int*) – Height of an image in the polar coordinate system.

Returns

- **x_mat** (*array_like*) – 2D array. Broadcast of the x-coordinates.
- **y_mat** (*array_like*) – 2D array. Broadcast of the y-coordinates.

algotom.util.utility.**separate_frequency_component**(*mat*, *axis=0*, *window={'name': 'gaussian', 'sigma': 5}*)
Separate low and high frequency components of an image along an axis. e.g axis=0 is to apply the separation to each column.

Parameters

- **mat** (*array_like*) – 2D array.
- **axis** (*int*) – Axis along which to apply the filter.
- **window** (*array_like or dict*) – 1D array or a dictionary which given the name of a window in the scipy.signal.window list and its parameters (without window-length).

Returns

- **mat_low** (*array_like*) – 2D array. Low-frequency image.
- **mat_high** (*array_like*) – 2D array. High-frequency image.

algotom.util.utility.**sort_backward**(*mat*, *mat_index*, *axis=0*)
Sort grayscales of an image using an index array provided. e.g axis=0 is to sort each column.

Parameters

- **mat** (*array_like*) – 2D array.
- **mat_index** (*array_like*) – 2D array. Index array used for sorting.
- **axis** (*int*) – Axis along which to sort.

Returns **mat_sort** (*array_like*) – 2D array. Sorted image.

algotom.util.utility.**sort_forward**(*mat*, *axis=0*)
Sort grayscales of an image along an axis. e.g. axis=0 is to sort along each column.

Parameters

- **mat** (*array_like*) – 2D array.
- **axis** (*int*) – Axis along which to sort.

Returns

- **mat_sort** (*array_like*) – 2D array. Sorted image.
- **mat_index** (*array_like*) – 2D array. Index array used for sorting backward.

algotom.util.utility.**transform_1d_window_to_2d**(*win_1d*)
Transform a 1d-window to 2d-window. Useful for designing a Fourier filter.

Parameters **win_1d** (*array_like*) – 1D array.

Returns **win_2d** (*array_like*) – Square array, a 2D version of the 1d-window.

algotom.util.utility.**transform_slice_backward**(*mat*, *coord_mat=None*)
Transform a reconstructed image in polar coordinates back to rectangular coordinates.

Parameters

- **mat** (*array_like*) – Square array. Reconstructed image in polar coordinates.
- **coord_mat** (*tuple of array_like, optional*) – (Square array of r-coordinates , square array of theta-coordinates) or generated if None.

Returns *array_like* – Transformed image.

algotom.util.utility.**transform_slice_forward**(*mat*, *coord_mat=None*)
Transform a reconstructed image into polar coordinates.

Parameters

- **mat** (*array_like*) – Square array. Reconstructed image.
- **coord_mat** (*tuple of array_like, optional*) – (Square array of x-coordinates , square array of y-coordinates) or generated if None.

Returns *array_like* – Transformed image.

1.6 Highlights

Algotom was used for some experiments featured on media:

- Scanning Moon rocks and Martian meteorites using helical scans with offset rotation-axis. Featured on Reuters.
- Scanning Herculaneum Scrolls using grid scans with offset rotation-axis respect to the grid's FOV. Featured on BBC.
- Scanning Little Foot fossil using two-camera detector with offset rotation-axis. Featured on BBC_.

1.7 Credits

1.7.1 Citations

We kindly request that you cite the following article [A1] if you use project.

1.7.2 References

BIBLIOGRAPHY

[A1] Nghia T. Vo, Robert C. Atwood, Michael Drakopoulos, and Thomas Connolley. Data processing methods and data acquisition for samples larger than the field of view in parallel-beam tomography. *Opt. Express*, 29(12):17849–17874, Jun 2021. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-29-12-17849, doi:10.1364/OE.418448.

[B1] Nghia T. Vo, Michael Drakopoulos, Robert C. Atwood, and Christina Reinhard. Reliable method for calculating the center of rotation in parallel-beam tomography. *Opt. Express*, 22(16):19078–19086, Aug 2014. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-22-16-19078, doi:10.1364/OE.22.019078.

PYTHON MODULE INDEX

a

INDEX

D

E

F

G

I

S

T

U

W