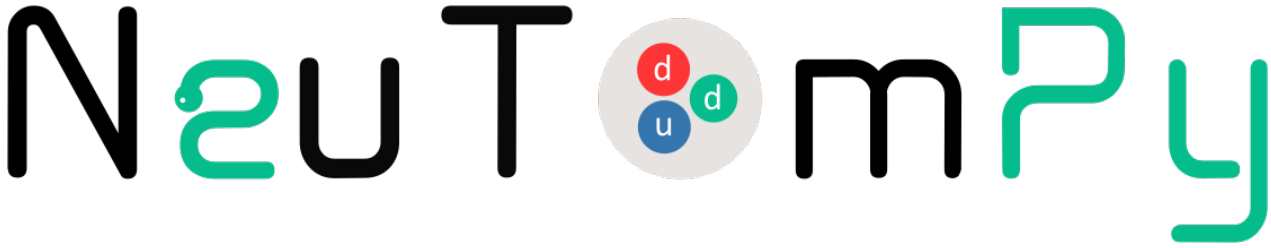
NeuTomPy toolbox Documentation

Release 1.0.10

Davide Micieli

Feb 19, 2020

Contents



NeuTomPy toolbox is a Python package for tomographic data processing and reconstruction. The toolbox includes pre-processing algorithms, artifacts removal and a wide range of iterative reconstruction methods as well as the Filtered Back Projection algorithm. The NeuTomPy toolbox was conceived primarily for Neutron Tomography and developed to support the need of users and researchers to compare state-of-the-art reconstruction methods and choose the optimal data-processing workflow for their data.

CHAPTER 1

Features

- Readers and writers for TIFF and FITS files and stack of images
- Data normalization with dose correction, correction of the rotation axis tilt, ring-filters, outlier removals, beam-hardening correction
- A wide range of reconstruction algorithms powered by ASTRA toolbox : FBP, SIRT, SART, ART, CGLS, NN-FBP, MR-FBP
- Image quality assessment with several metrics

CHAPTER 2

Reference

If you use the NeuTomPy toolbox for your research, please cite the following paper:

- D. Micieli, T. Minniti, G. Gorini, "NeuTomPy toolbox, a Python package for tomographic data processing and reconstruction", SoftwareX, Volume 9 (2019), pp. 260-264, https://doi.org/10.1016/j.softx.2019.01.005.

CHAPTER 3

Table of Contents

3.1 Installation

NeuTomPy toolbox supports **Linux**, **Windows** and **Mac OS** 64-bit operating systems.

First of all, install a Conda python environment with **Python 3.5 or 3.6**.

It is required to install some dependencies, hence run the following inside a `conda` environment:

```
conda install -c simpleitk simpleitk
conda install -c astra-toolbox astra-toolbox
conda install -c conda-forge ipython numpy numexpr matplotlib astropy tifffile opencv␣
↪scikit-image read-roi mkl_fft scipy six tqdm pywavelets
```

Then install NeuTomPy toolbox via `pip`:

```
pip install neutompy
```

NB: If a segmentation fault occurs when importing NeuTomPy, install PyQt5 via `pip`:

```
pip install PyQt5
```

3.1.1 Update

To update a NeuTomPy installation to the latest version run:

```
pip install neutompy --upgrade
```

3.2 Examples

This section includes some example scripts.

3.2.1 FBP reconstruction with GPU

```
# ----------------------------------------------------------------------
# This script performs a complete reconstruction workflow.
# The reconstruction algorithm used is the FBP performed on a GPU.
# ----------------------------------------------------------------------
import numpy as np
import neutompy as ntp

# set pixel size in cm
pixel_size  = 0.0029

# set the last angle value of the CT scan: np.pi or 2*np.pi
last_angle = 2*np.pi

# read dataset containg projection, dark-field, flat-field images and the projection at 180 degree
proj, dark, flat, proj_180 = ntp.read_dataset()

# normalize the projections to dark-field, flat-field images and neutron dose
norm, norm_180 = ntp.normalize_proj(proj, dark, flat, proj_180=proj_180, dose_draw=True, crop_draw=True)

# rotation axis tilt correction
norm = ntp.correction_COR(norm, norm[0], norm_180)

# clean up memory
del dark; del flat; del proj; del proj_180

# remove outliers, set the optimal radius and threshold
norm = ntp.remove_outliers_stack(norm, radius=1, threshold=0.018, outliers='dark', out=norm)
norm = ntp.remove_outliers_stack(norm, radius=3, threshold=0.018, outliers='bright', out=norm)

# perform minus-log transform
norm =  ntp.log_transform(norm, out=norm)

# remove stripes in sinograms
norm = ntp.remove_stripe_stack(norm, level=4, wname='db30', sigma=1.5, out=norm)

# define the array of the angle views in radians
angles = np.linspace(0, last_angle, norm.shape[0], endpoint=False)

# FBP reconstruction with the hamming filter using GPU
print('> Reconstruction...')
rec    = ntp.reconstruct(norm, angles, 'FBP_CUDA', parameters={"FilterType":"hamming"}, pixel_size=pixel_size)

#    Implemented FilterType in ASTRA toolbox are:
#         ``ram-lak`` (default), ``shepp-logan``, ``cosine``, ``hamming``, ``hann``, ``none``, ``tukey``,
#         ``lanczos``, ``triangular``, ``gaussian``, ``barlett-hann``, ``blackman``, ``nuttall``,
#         ``blackman-harris``, ``blackman-nuttall``, ``flat-top``, ``kaiser``,
#         ``parzen``, ``projection``, ``sinogram``, ``rprojection``, ``rsinogram``.

```

(continues on next page)

(continued from previous page)

```

# select the directory and the prefix file name of the reconstructed images to save.
recon_dir = ntp.save_filename_gui('', message = 'Select the folder and the prefix␣
↪name for the reconstructed images...')

# write the reconstructed images to disk
ntp.write_tiff_stack(recon_dir, rec)
```

3.2.2 FBP reconstruction with CPU

```
# ----------------------------------------------------------------------
# This script performs a complete reconstruction workflow.
# The reconstruction algorithm used is the FBP performed on CPU.
# ----------------------------------------------------------------------
import numpy as np
import neutompy as ntp

# set pixel size in cm
pixel_size  = 0.0029

# set the last angle value of the CT scan: np.pi or 2*np.pi
last_angle = 2*np.pi

# read dataset containg projection, dark-field, flat-field images and the projection␣
↪at 180 degree
proj, dark, flat, proj_180 = ntp.read_dataset()

# normalize the projections to dark-field, flat-field images and neutron dose
norm, norm_180 = ntp.normalize_proj(proj, dark, flat, proj_180=proj_180, dose_
↪draw=True, crop_draw=True)

# rotation axis tilt correction
norm = ntp.correction_COR(norm, norm[0], norm_180)

# clean up memory
del dark; del flat; del proj; del proj_180

# remove outliers, set the optimal radius and threshold
norm = ntp.remove_outliers_stack(norm, radius=1, threshold=0.018, outliers='dark',␣
↪out=norm)
norm = ntp.remove_outliers_stack(norm, radius=3, threshold=0.018, outliers='bright',␣
↪out=norm)

# perform minus-log transform
norm =  ntp.log_transform(norm, out=norm)

# remove stripes in sinograms
norm = ntp.remove_stripe_stack(norm, level=4, wname='db30', sigma=1.5, out=norm)

# define the array of the angle views in radians
angles = np.linspace(0, last_angle, norm.shape[0], endpoint=False)

# FBP reconstruction using CPU
print('> Reconstruction...')
rec    = ntp.reconstruct(norm, angles, 'FBP', pixel_size=pixel_size)
```

(continues on next page)

(continued from previous page)

```

# select the directory and the prefix file name of the reconstructed images to save.
recon_dir = ntp.save_filename_gui('', message = 'Select the folder and the prefix name for the reconstructed images...')

# write the reconstructed images to disk
ntp.write_tiff_stack(recon_dir, rec)
```

3.2.3 SIRT reconstruction with GPU

```
# ----------------------------------------------------------------------
# This script performs a complete reconstruction workflow.
# The reconstruction algorithm used is the SIRT performed on a GPU.
# ----------------------------------------------------------------------
import numpy as np
import neutompy as ntp

# set pixel size in cm
pixel_size  = 0.0029

# set the last angle value of the CT scan: np.pi or 2*np.pi
last_angle = 2*np.pi

# read dataset containg projection, dark-field, flat-field images and the projection at 180 degree
proj, dark, flat, proj_180 = ntp.read_dataset()

# normalize the projections to dark-field, flat-field images and neutron dose
norm, norm_180 = ntp.normalize_proj(proj, dark, flat, proj_180=proj_180, dose_draw=True, crop_draw=True)

# rotation axis tilt correction
norm = ntp.correction_COR(norm, norm[0], norm_180)

# clean up memory
del dark; del flat; del proj; del proj_180

# remove outliers, set the optimal radius and threshold
norm = ntp.remove_outliers_stack(norm, radius=1, threshold=0.018, outliers='dark', out=norm)
norm = ntp.remove_outliers_stack(norm, radius=3, threshold=0.018, outliers='bright', out=norm)

# perform minus-log transform
norm =  ntp.log_transform(norm, out=norm)

# remove stripes in sinograms
norm = ntp.remove_stripe_stack(norm, level=4, wname='db30', sigma=1.5, out=norm)

# define the array of the angle views in radians
angles = np.linspace(0, last_angle, norm.shape[0], endpoint=False)

# SIRT reconstruction with 100 iterations using GPU
print('> Reconstruction...')
rec    = ntp.reconstruct(norm, angles, 'SIRT_CUDA', parameters={"iterations":100}, pixel_size=pixel_size)
```

(continues on next page)

(continued from previous page)

```

# select the directory and the prefix file name of the reconstructed images to save.
recon_dir = ntp.save_filename_gui('', message = 'Select the folder and the prefix name for the reconstructed images...')

# write the reconstructed images to disk
ntp.write_tiff_stack(recon_dir, rec)
```

3.2.4 NN-FBP reconstruction

```
# ----------------------------------------------------------------------
# This script shows an usage example of the NN-FBP method.
# A complete dataset is reconstructed via FBP and the NN-FBP is
# trained to reconstruct some reconstructed slices using a sparse-view
# dataset. Then different slices are reconstructed via NN-FBP.
# ----------------------------------------------------------------------

import numpy as np
import neutompy as ntp
import os

# set pixel size in cm
pixel_size  = 0.0029

hqrec_folder = 'hqrecs/'  # folder to save high quality reconstuction
nnfbp_rec_folder = 'recon-nnfbp/' # output folder of the nn-fbp reconstruction
conf = {}
conf['hidden_nodes']  = 3 # number of hidden nodes
conf['hqrecfiles']    = hqrec_folder + 'sample*.tiff' # high-quality reconstruction files
conf['traindir']      = 'trainfiles/'  # folder where training files are stored
conf['npick']         = 10000          # number of random pixels to pick per slice
conf['filter_file']   = 'filters.mat'  # file to store trained filters

# set the last angle value of the CT scan: np.pi or 2*np.pi
last_angle = 2*np.pi

# read dataset containing projection, dark-field, flat-field images and the projection at 180 degree
proj, dark, flat, proj_180 = ntp.read_dataset()

# normalize the projections to dark-field, flat-field images and neutron dose
norm, norm_180 = ntp.normalize_proj(proj, dark, flat, proj_180=proj_180, dose_draw=True, crop_draw=True, log=True)

# rotation axis tilt correction
norm = ntp.correction_COR(norm, norm[0], norm_180)

# define the array of the angle views in radians
angles = np.linspace(0, last_angle, norm.shape[0], endpoint=False)

# high-quality reconstruction
train_slice_start = 100
train_slice_end  = 120
rec = ntp.reconstruct(norm[:,train_slice_start:train_slice_end+1, :], angles, 'FBP_CUDA', parameters={"FilterType":"hamming"}, pixel_size=pixel_size)
```

(continues on next page)

(continued from previous page)

```

# write the high-quality reconstructed images to disk
try:
        os.mkdir(hqrec_folder)
except OSError:
        pass
ntp.write_tiff_stack(hqrec_folder + 'sample', rec)

# NN-FBP training
skip  = 3  # reduction factor of the full dataset to obtain the sparse-view dataset
norm_train = norm[::skip,train_slice_start:train_slice_end+1, :]
ntp.reconstruct(norm_train, angles[::skip], 'NN-FBP-train', parameters=conf)

# NN-FBP reconstruction of noisy projections
test_slice_start = 180
test_slice_end   = 200
norm_test = norm[::skip,test_slice_start:test_slice_end+1, :]
rec_nnfbp = ntp.reconstruct(norm_test, angles[::skip], 'NN-FBP', parameters=conf)

# write NN-FBP reconstructed images
try:
        os.mkdir(nnfbp_rec_folder)
except OSError:
        pass
ntp.write_tiff_stack(nnfbp_rec_folder + 'sample', rec_nnfbp)
```

3.2.5 Reconstruction quality assessment

```
# ----------------------------------------------------------------------
# This script performs FBP, SIRT and CGLS reconstructions of a
# phantom sample. The reconstructions are compared using several
# image quality indexes.
# ----------------------------------------------------------------------

import numpy as np
import neutompy as ntp
from matplotlib.gridspec import GridSpec
import matplotlib.pyplot as plt

# pixel size in cm
pixel_size = 0.0029

# factor to reduce the number of projections in the sinogram
skip_theta = 3

# image filename
fname = './data/sinogram.tiff'

# read the sinogram
sino_hq = ntp.read_image(fname)
na      = sino_hq.shape[0]

# reduce number of projections in the sinogram
sino    = sino_hq[::skip_theta]

```

(continues on next page)

(continued from previous page)

```
# angles views in radians
angles = np.linspace(0, 2*np.pi, na, False)[::skip_theta]

# ground truth reconstruction
true = ntp.reconstruct(sino_hq, np.linspace(0, np.pi*2, sino_hq.shape[0], endpoint=False), 'SIRT_CUDA',
                                                    {'iterations':200}, pixel_size=pixel_size)
# fbp reconstruction
fbp  = ntp.reconstruct(sino, angles, 'FBP_CUDA', pixel_size=pixel_size)
# sirt reconstruction
sirt = ntp.reconstruct(sino, angles, 'SIRT_CUDA', parameters={'iterations':200}, pixel_size=pixel_size)
# cgls reconstruction
cgls = ntp.reconstruct(sino, angles, 'CGLS_CUDA', parameters={'iterations':10}, pixel_size=pixel_size)

# define a list of reconstructed images
rec_list = [fbp, sirt, cgls]
rec_name = ['FBP', 'SIRT', 'CGLS']

# roi coordinates
rmin = 0
rmax = None
cmin = 0
cmax = 880

# set the x-axis range of the histograms
xmin = 0.0
xmax = 0.7

# counts range of the histograms
ymin = 10
ymax = 2e4

nsquare = 3
nbins = 300

[binning, width] = np.linspace(xmin, xmax, nbins, retstep=True)

nsubplot = len(rec_list)

plt.rc('font', family='serif', serif='Times', size=11)
plt.rc('text', usetex=False)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)
plt.rc('axes', labelsize=12)


fig = plt.figure(figsize=(nsquare*nsubplot,nsquare+1+0.5))
fig.subplots_adjust(hspace=0, wspace=0.5, top=0.8)
gs = GridSpec(nsquare+1,nsquare*nsubplot)

for i in range(0, nsubplot):

        ax1=fig.add_subplot(gs[0:nsquare,i*nsquare:(i+1)*nsquare])
        # quality metrics evaluation
```

(continues on next page)

(continued from previous page)

```
        img = rec_list[i]
        im = ax1.imshow(img[rmin:rmax, cmin:cmax], vmin=xmin, vmax=xmax, cmap='gray')
        ssim  = ntp.SSIM(img, true)
        nrmse = ntp.NRMSE(img, true)
        cnr   = ntp.CNR(img, froi_signal='./data/signal.roi', froi_background='./data/
↪background.roi')

        title = rec_name[i]
        plt.title(title +  '\n SSIM = '+ "{:.2f}".format(ssim) + ', CNR = ' "{:.1f}".
↪format(cnr) + ',\n NRMSE = '+ "{:.2f}".format(nrmse))
        plt.xticks([])
        plt.yticks([])

        ax2=fig.add_subplot(gs[nsquare,i*nsquare:(i+1)*nsquare])
        # generate histogram of the gray values inside a circular mask
        mask = ntp.get_circular_mask(img.shape[0], img.shape[1], radius=370,␣
↪center=(img.shape[0]//2, img.shape[0]//2 -30))
        cc, edge = np.histogram(img[mask], bins=binning)
        ax2.bar(edge[:-1]+width*0.5, cc, width, color='C3', edgecolor='C3', log=True)
        plt.xlim([0, xmax])
        plt.ylim([ymin, ymax])
        plt.yticks([1e2, 1e3, 1e4], ['']*3)
        if i == 0:
                plt.yticks([1e2, 1e3, 1e4], ['$10^2$', '$10^3$', '$10^4$'])


plt.show()
```

A sample dataset for testing purpose can be found here . This dataset includes neutron radiographs of a phantom sample acquired at the IMAT beamline, ISIS neutron spallation source, UK.

3.3 API reference

This section contains the API reference for NeuTomPy toolbox.

NeuTomPy Modules:

3.3.1 neutompy.image.image

`neutompy.image.image.`**`read_tiff`**(*fname*, *croi=None*, *froi=None*)

This function reads a 2D TIFF image.

Parameters

- **fname** (*str*) – String defining the file name or the file path
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI file name or file path.

Returns **out** (*ndarray*) – The 2D image. If the file specified doesn't exist, the function returns False.

`neutompy.image.image.`**`read_fits`**(*fname*, *croi=None*, *froi=None*)

This function reads a 2D FITS image.

Parameters

- **fname** (*str*) – String defining the file name or the file path
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI filename or file path.

Returns **out** (*ndarray*) – The 2D image. If the file specified doesn't exist, the function returns False.

neutompy.image.image.**read_fits_stack**(*fname*, *slices=[]*, *croi=None*, *froi=None*)

Read stack of fits images.

Parameters

- **fname** (*str*) – One of the file names of the fits stack. File with unknown image extension are skipped automatically. If fname is '' or a folder path, then a dialog box is opened to select one of the file of the stack to read. The initial directory is C:(in windows) or / (in UNIX) if fname==''', otherwise is the folder path assigned to fname.
- **slices** (*list of int, optional*) – List of the element indexes to read. If it is [] then all fits image files are read.
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI filename or file path.

Returns **array** (*ndarray*) – The 3D stack of images

neutompy.image.image.**read_tiff_stack**(*fname*, *slices=[]*, *croi=None*, *froi=None*)

Read stack of tiff images.

Parameters

- **fname** (*str*) – One of the file names of the tiff stack. File with unknown image extension are skipped automatically. If fname is '' or a folder path, then a dialog box is opened to select one of the file of the stack to read. The initial directory is C:(in windows) or / (in UNIX) if fname==''', otherwise is the folder path assigned to fname.
- **slices** (*list of int, optional*) – List of the element indexes to read. If it is [] then all tif image files are read.
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI filename or file path.

Returns **array** (*ndarray*) – The 3D stack of images

neutompy.image.image.**get_rect_coordinates_from_roi**(*fname*)

This function returns the coordinates from a rectangular region of interest defined in a .roi file generated by ImageJ / Fiji.

N.B.: indexing starts from 0. Incremet rowmax and col_max + 1 to crop an image, for example: crop = img[rowmin:(rowmax+1), colmin:(colmax+1)]

Parameters **fname** (*str*) – String defining the path or the name of the Image ROI file.

Returns

- **rowmin** (*int*) – The minimum row coordinate.
- **rowmax** (*int*) – The maximum row coordinate.

- **colmin** (*int*) – The minimum column coordinate.
- **colmax** (*int*) – The maximum column coordinate.

neutompy.image.image.**read_image**(*fname*, *croi=None*, *froi=None*)
This function reads a 2D TIF or FITS image.

Parameters

- **fname** (*str*) – String defining the file name or the file path
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI filename or file path.

Returns **out** (*ndarray*) – The 2D image. If the file specified doesn't exist, the function returns False.

neutompy.image.image.**read_stack_from_list**(*flist*, *slices=[]*, *croi=None*, *froi=None*)
Read stack of images from file list.

Parameters

- **flist** (*list of str*) – List of the file names or file paths to read in the defined order. File with unknown image extension are skipped automatically.
- **slices** (*list of int, optional*) – List of the indexes of flist to read. If it is [] then all image files are read.
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI filename or file path.

Returns **array** (*ndarray*) – The 3D stack of images

neutompy.image.image.**read_dataset**(*proj_180=True*, *croi=None*, *froi=None*)
This function reads a dataset which contains dark-field, flat-field, projection images and optionally the projection at 180 degree. The user selects the main folder and the files from a dialog box. A two-dimensional region of interest of each image can be read specifying the coordinates or an ImageJ .roi file.

Parameters

- **proj_180** (*bool, optional*) – If True the user must select the projection at 180 degree separately from the stack of projections.
- **croi** (*tuple, optional*) – Tuple defining the ROI indexes for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI file name or file path.

Returns

- **proj** (*3d array*) – The array containing the stack of projections.
- **dark** (*3d array*) – The array containing the stack of dark-field images.
- **flat** (*3d array*) – The array containing the stack of flat-field images.
- **proj180** (*2d array*) – Only if proj_180 is True, the 2D array representing the projection at 180 degree is returned separately.

neutompy.image.image.**read_image_stack**(*fname*, *slices=[]*, *croi=None*, *froi=None*)
Read stack of TIFF or FITS images. This function recognize the file type from the file extension.

Parameters

- **fname** (*str*) – One of the file names of the tiff stack. File with unknown image extension are skipped automatically.
- **slices** (*list of int, optional*) – List of the element indexes to read. If it is [] then all tif image files are read.
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **froi** (*str, optional*) – String defining the ImageJ ROI filename or file path.

Returns **array** (*ndarray*) – The 3D stack of images

`neutompy.image.image.`**`write_fits`**(*fname*, *img*, *overwrite=False*)

This function write to the disk an array as fits image.

Parameters

- **fname** (*str*) – String defining the file name or file path of the image to save. If the extension is not specified, it is automatically appended to fname.
- **img** (*ndarray*) – The array to save as image.
- **overwrite** (*bool, optional*) – If `True`, overwrites the output file if it exists. Raises an `IOError` if `False` and the output file exists. Default is `False`.

`neutompy.image.image.`**`write_tiff`**(*fname*, *img*, *overwrite=False*)

This function write to the disk an array as tiff image.

Parameters

- **fname** (*str*) – String defining the file name or file path of the image to save. If the extension is not specified, it is automatically appended to fname.
- **img** (*ndarray*) – The array to save as image.
- **overwrite** (*bool, optional*) – If `True`, overwrites the output file if it exists. Raises an `IOError` if `False` and the output file exists. Default is `False`.

`neutompy.image.image.`**`write_tiff_stack`**(*fname*, *data*, *axis=0*, *start=0*, *croi=None*, *digit=4*, *dtype=None*, *overwrite=False*)

This function writes a 3D array to a stack of 2D TIFF images.

Parameters

- **fname** (*str*) – String defining the prefix of the file name and the containing folder.
- **data** (*ndarray*) – The 3D stack to write.
- **axis** (*int, optional*) – The axis along which the stacking is performed.
- **start** (*int, optional*) – Index used for saving the first image.
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **digit** (*int, optional*) – Number of digits used for the numbering of the images.
- **dtype** (*dtype, optional*) – Data type of the images to save.
- **overwrite** (*bool, optional*) – If `True`, overwrites the output file if it exists. Raises an `IOError` if `False` and the output file exists. Default is `False`.

`neutompy.image.image.`**`write_fits_stack`**(*fname*, *data*, *axis=0*, *start=0*, *croi=None*, *digit=4*, *dtype=None*, *overwrite=False*)

This function writes a 3D array to a stack of 2D FITS images.

Parameters

- **fname** (*str*) – String defining the prefix of the file name and the containing folder.
- **data** (*ndarray*) – The 3D stack to write.
- **axis** (*int, optional*) – The axis along which the stacking is performed.
- **start** (*int, optional*) – Index used for saving the first image.
- **croi** (*tuple, optional*) – Tuple defining the indexes range for each axis. It must be follow this notation: ((row_start, row_end, row_step), (col_start, col_end, col_step))
- **digit** (*int, optional*) – Number of digits used for the numbering of the images.
- **dtype** (*dtype, optional*) – Data type of the images to save.
- **overwrite** (*bool, optional*) – If `True`, overwrites the output file if it exists. Raises an `IOError` if `False` and the output file exists. Default is `False`.

`neutompy.image.image.`**`get_rect_coordinates_from_roi`**(*fname*)

This function returns the coordinates from a rectangular region of interest defined in a .roi file generated by ImageJ / Fiji.

N.B.: indexing starts from 0. Incremet rowmax and col_max + 1 to crop an image, for example: crop = img[rowmin:(rowmax+1), colmin:(colmax+1)]

Parameters **fname** (*str*) – String defining the path or the name of the Image ROI file.

Returns

- **rowmin** (*int*) – The minimum row coordinate.
- **rowmax** (*int*) – The maximum row coordinate.
- **colmin** (*int*) – The minimum column coordinate.
- **colmax** (*int*) – The maximum column coordinate.

`neutompy.image.image.`**`get_filename_pattern`**(*fname*)

3.3.2 neutompy.image.rebin

`neutompy.image.rebin.`**`rebin`**(*arr*, *binsize*, *dtype=<class 'numpy.float32'>*)

This function rebins a single 2D image or 3D stack. It reduces the size of an image or stack of images by binning groups of pixels of user-specified sizes. The resulting pixels are computed as average.

Parameters

- **arr** (*ndarray, 2d or 3d*) – The 2D image or the 3D stack of images to rebin.
- **binsize** (*tuple*) – Tuple defining the bin size for each axis of the array. E.g.: for 2d array (bin_0, bin_1)

  for 3d array (bin_0, bin_1, bin_2)
- **dtype** (*type, optional*) – The type of the returned rebbined array.

Returns **m** (*ndarray*) – The rebinned image or stack of images.

3.3.3 neutompy.preproc.preproc

neutompy.preproc.preproc.**draw_ROI**(*img*, *title*, *ratio=0.85*)

This function allows to select interactively a rectangular region of interest (ROI) over an image. The function returns the ROI coordinates.

Parameters

- **img** (*2d array*) – The image on which the dose roi is drawn.
- **title** (*str*) – String defining the title of the window shown.
- **ratio** (*float, optional*) – The filling ratio of the window respect to the screen resolution. It must be a number between 0 and 1. The default value is 0.85.

Returns

- **rowmin** (*int*) – The minimum row coordinate.
- **rowmax** (*int*) – The maximum row coordinate.
- **colmin** (*int*) – The minimum column coordinate.
- **colmax** (*int*) – The maximum column coordinate.

neutompy.preproc.preproc.**normalize_proj**(*proj*, *dark*, *flat*, *proj_180=None*, *out=None*, *dose_file=''*, *dose_coor=()*, *dose_draw=True*, *crop_file=''*, *crop_coor=()*, *crop_draw=True*, *scattering_bias=0.0*, *minus_log_lowest_val=None*, *min_denom=1e-06*, *min_ratio=1e-06*, *max_ratio=10.0*, *mode='mean'*, *log=False*, *sino_order=False*, *show_opt='mean'*)

This function computes the normalization of the projection data using dark and flat images. If the source intensity is not stable the images can be normalized respect to the radiation dose (see formula 2.2 in[1]). In this case, the user must specify a region (the dose ROI) where the sample never appears. If not interested in reconstructing the entire field-of-view, the normalization can be performed using only a region of interest (crop ROI) of all projections. The dose ROI and the crop ROI can be drawn interactively on the image or specified using the index coordinates or an ImageJ .roi file. A stack of dark and flat images is required and the median or the mean of the stack is used to compute the main fomula.

Parameters

- **proj** (*ndarray*) – A three-dimensional stack of raw projections. The 0-axis represents theta.
- **dark** (*ndarray*) – A three-dimensional stack of dark-field images.
- **flat** (*ndarray*) – A three-dimensional stack of flat-field images.
- **proj_180** (*2d arrays, optional*) – The projection at 180 degree. Specify it only if it is not already included in the stack *proj*. It is disabled by default (None).
- **out** (*ndarray, optional*) – The output array returned by the function. If it is the same as proj, the computation will be done in place.
- **dose_file** (*str, optional*) – String defining the path or the name of the Image ROI file that includes the dose ROI.
- **dose_coor** (*tuple, optional*) – Tuple defining the indexes range for each axis of the ROI dose. Specify it in this way: `(row_start, row_end, col_start, col_end)`

[1] D. Micieli et al., A comparative analysis of reconstruction methods applied to Neutron Tomography, Journal of Instrumentation, Volume 13, June 2018.

- **dose_draw** (*bool, optional*) – If True the dose ROI is selected interactively on the image by the user. The default is True.
- **crop_file** (*str, optional*) – String defining the path or the name of the Image ROI file that includes the ROI to crop.
- **crop_coor** (*tuple, optional*) – Tuple defining the indexes range, for each axis, of the ROI to crop. Specify it in this way: `(row_start, row_end, col_start, col_end)`
- **crop_draw** (*bool, optional*) – If True the ROI to crop is selected interactively on the image by the user. The default is True.
- **scattering_bias** (*float, optional*) – It allows to compensate uniform scattering by subtracting a constant value from the raw projections. Defalut value is `0.0`.
- **minus_log_lowest_val** (*float, optional*) – Minimum permitted value of the -log-transform of the normalized data. The parameter prevents values close to (or below) zero to introduce artefacts. The clipping is ignored if minus_log_lowest_val is None or the variable log is `False`. Default value is `None`.
- **min_denom** (*float, optional*) – Minimum permitted value of the denominator. It must be a small number that prevents the division by zero. Defalut value is `1.0e-6`.
- **min_ratio** (*float, optional*) – Minimum permitted value of normalized projections. It must be a small positive number that prevents negative values normalized data. Defalut value is `1.0e-6`.
- **max_ratio** (*float, optional*) – Maximum permitted value of normalized projections. It must be a positive number. It mitigates the magnitude of the bright outliers within normalized data. Defalut value is `10`.
- **log** (*bool, optional*) – If `True` the log-transform of the normalized data is performed. If `False`, the normalized data without log-transform are returned. Default value is `False`.
- **mode** (*string, optional*) – If *dose_draw* or *crop_draw* is `True` the user can select interactively the ROI. A window showing a representative image of the projection stack is created. This image can be the mean or the standard deviation computed pixel-wise over the projection stack. Hence, allowed values of *mode* are `mean` and `std`. Default value is `mean`.
- **sino_order** (*bool, optional*) – If `True` a stack of sinograms is returned (0 axis represents the projections y-axis). If `False` a stack of projections is returned (0 axis represents theta). Default value is `False`.

Returns

- **out** (*ndarray*) – Three-dimensional stack of the normalized projections.
- **out_180** (*2d arrays*) – The normalized projection at 180 degree. It is returned only if *proj_180* is an array.

References

Examples

Normalize dataset selecting interactively the ROI to crop and the dose ROI.

```
>>> import neutompy as ntp
>>> norm =  ntp.normalize_proj(proj, dark, flat, dose_draw=True, crop_draw=True)
```

Normalize dataset and the raw projection at 180 degree:

```
>>> fname  = ntp.get_image_gui('', message = 'Select raw projection at 180°...')
>>> img180 = ntp.read_image(fname)
>>> norm, norm_180 = ntp.normalize_proj(proj, dark, flat, proj_180=img180)
```

Normalize dataset using two ImageJ .roi file to define the ROI to crop and the dose ROI:

```
>>> norm = ntp.normalize_proj(proj, dark, flat, dose_file='./dose.roi', crop_file=
↪'./crop.roi'
                                                  dose_draw=False, crop_draw=False)
```

Normalize the dataset with the log-transform:

```
>>> norm = ntp.normalize_proj(proj, dark, flat, log=True)
```

Trivial data normalization using the whole field of view and without the dose correction:

```
>>> norm = ntp.normalize_proj(proj, dark, flat, dose_draw=False, crop_draw=False)
```

neutompy.preproc.preproc.**log_transform**(*norm_proj*, *out=None*)

This function computes the minus log of an array. In trasmission CT the input array must be the normalized dataset after flat-fielding correction.

Parameters

- **norm_proj** (*ndarray*) – 3D stack of projections.
- **out** (*ndarray, optional*) – Output array. If same as norm_proj, computation will be done in-place.

Returns **out** (*ndarray*) – Minus-log of the input array.

neutompy.preproc.preproc.**find_COR**(*proj_0*, *proj_180*, *nroi=None*, *ref_proj=None*, *ystep=5*, *ShowResults=True*)

This function estimates the offset and the tilt angle of the rotation axis respect to the detector using the projections at 0 and at 180 degree. The user selects interactively different regions where the sample is visible. Once the position of the rotation axis is found the results are shown in two figures. In the first are shown the computed rotation axis and the image obtained as proj_0 - pro_180[:,::-1] (the projection at 180 is flipped horizontally) before the correction. The second figure shows the difference image proj_0 - pro_180[:,::-1] after the correction and the histogram of abs(proj_0 - pro_180[:,::-1]).

Parameters

- **proj_0** (*2d array*) – The projection at 0 degrees.
- **proj_180** (*2d array*) – The projection at 180 degrees.
- **nroi** (*int, optional*) – The number of the region of interest to select for the computation of the rotation axis position. Default is None, hence the value is read as input from keyboard.
- **ref_proj** (*2d array*) – The image shown to select the region of interest. Default is None, hence proj_0 is shown.
- **ystep** (*int, optional*) – The center of rotation position is computed every *ystep*. Default value is 5.
- **ShowResults** (*bool, optional*) – If True, the the two figures that summarize the result are shown. Default is True.

Returns

- **middle_shift** (*float*) – The horizontal shift of the rotation axis respect to the center of the detector.

- **theta** (*float*) – The tilt angle formed by the rotation axis and the vertical axis of the detector.

neutompy.preproc.preproc.**correction_COR**(*norm_proj*, *proj_0*, *proj_180*, *show_opt='mean'*, *shift=None*, *theta=None*, *nroi=None*, *ystep=5*)

This function corrects the misalignment of the rotation axis respect to the vertical axis of the detector. The user can choose to insert manually the offset and the tilt angle of the rotation axis respect to the detector, if known parameters, or to estimate them using the projections at 0 and at 180 degrees. In this case, the user selects interactively different regions where the sample is visible. Once the position of the rotation axis is found the results are shown in two figures. In the first are shown the estimated rotation axis and the image obtained as proj_0 - pro_180[:,::-1] (projection at 180 is flipped horizontally) before the correction. The second figure shows the difference image (proj_0 - pro_180[:,::-1]) after the correction and the histogram of abs(proj_0 - pro_180[:,::-1]).

Parameters

- **norm_proj** (*ndarray*) – A stack of projections. The 0-axis represents theta.
- **proj_0** (*2d array*) – The projection at 0 degrees.
- **proj_180** (*2d array*) – The projection at 180 degrees.
- **show_opt** (*str, optional*) – A string defining the image to show for the selection of the ROIs.

  Allowed values are:

  `mean` -> shows the mean of *norm_proj* along the O-axis.

  `std` -> shows the standard deviation of *norm_proj* along the O-axis

  `zero` -> shows proj_0

  `pi` -> shows proj_180

  Default value is `mean`.
- **shift** (*int, optional*) – The horizontal shift in pixel of the rotation axis respect to the vertical axis of the detector. It can be specified if known parameter. The default is None, hence this parameter is estimated from the projections.
- **theta** (*float, optional*) – The tilt angle in degrees of the rotation axis respect to the vertical axis of the detector. It can be specified if known parameter. The default is None, hence this parameter is estimated from the projections.
- **nroi** (*int, optional*) – The number of the region of interest to select for the computation of the rotation axis position. Default is None, hence the value is read as input from keyboard.
- **ystep** (*int, optional*) – The center of rotation position is computed every *ystep*. Default value is 5.

Returns **norm_proj** (*ndarray*) – The stack after the correction. The computation is done in place.

neutompy.preproc.preproc.**remove_outliers**(*img*, *radius*, *threshold*, *outliers='bright'*, *k=1.0*, *out=None*)

This function removes bright and dark outliers from an image. It replaces a pixel by the median of the pixels in the neighborhood if it deviates from the median by more than a certain value (k*threshold). The threshold can be specified by the user as a global value or computed proportionally to the local standard deviation of the projection.

Parameters

- **img** (*2d array*) – The image to elaborate.
- **radius** (*int or tuple of int*) – The radius of the neighborhood. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel.

- **threshold** (*float or str*) – If it is the string 'local' the local standard deviation map is taken into account. Conversely, if it is a float number the threshold is global.
- **outliers** (*str, optional*) – A string defining the type of outliers to remove. Allowed values are `bright` and `dark`. Default is *bright*.
- **k** (*float, optional*) – A pixel is replaced by the median of the pixels in the neighborhood if it deviates from the median by more than k*threshold. Default value is 1.0.
- **out** (*2d array, optional*) – If same as img, then the computation is done in place. Default is None, hence this behaviour is disabled.

Returns **out** (*2d array*) – The image obtained by removing the outliers.

`neutompy.preproc.preproc.`**`remove_outliers_stack`**(*arr*, *radius*, *threshold*, *axis=0*, *outliers='bright'*, *k=1.0*, *out=None*)

This function removes bright and dark outliers from a stack of images. The algorithm elaborates 2d images and the filtering is iterated over all images in the stack. The function replaces a pixel by the median of the pixels in the 2d neighborhood if it deviates from the median by more than a certain value (k*threshold). The threshold can be specified by the user as a global value or computed proportionally to the local standard deviation of the projection.

Parameters

- **arr** (*ndarray*) – The stack to elaborate.
- **radius** (*int or tuple of int*) – The radius of the 2D neighborhood. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel.
- **threshold** (*float or str*) – If it is the string 'local' the local standard deviation map is taken into account. Conversely, if it is a float number the threshold is global.
- **axis** (*int*) – The axis along wich the outlier removal is iterated.
- **outliers** (*str, optional*) – A string defining the type of outliers to remove. Allowed values are `bright` and `dark`. Default is *bright*.
- **k** (*float, optional*) – A pixel is replaced by the median of the pixels in the neighborhood if it deviates from the median by more than k*threshold. Default value is 1.0.
- **out** (*2d array, optional*) – If same as arr, then the computation is done in place. Default is None, hence this behaviour is disabled.

Returns **out** (*ndarray*) – The array obtained by removing the outliers.

`neutompy.preproc.preproc.`**`remove_stripe`**(*img*, *level*, *wname='db5'*, *sigma=1.5*)

Suppress horizontal stripe in a sinogram using the Fourier-Wavelet based method by Munch et al.[2].

Parameters

- **img** (*2d array*) – The two-dimensional array representig the image or the sinogram to destripe.
- **level** (*int*) – The highest decomposition level.
- **wname** (*str, optional*) – The wavelet type. Default value is `db5`
- **sigma** (*float, optional*) – The damping factor in the Fourier space. Default value is `1.5`

Returns **out** (*2d array*) – The resulting filtered image.

[2] B. Munch, P. Trtik, F. Marone, M. Stampanoni, Stripe and ring artifact removal with combined wavelet-Fourier filtering, Optics Express 17(10):8567-8591, 2009.

References

neutompy.preproc.preproc.**remove_stripe_stack**(*arr*, *level*, *wname='db5'*, *sigma=1.5*, *axis=1*, *out=None*)

Suppress horizontal stripe in a stack of sinograms or a stack of projections using the Fourier-Wavelet based method by Munch et al.[3] .

Parameters

- **arr** (*3d array*) – The tomographic data. It can be a stack of projections (theta is the 0-axis) or a stack of images (theta is the 1-axis).
- **level** (*int*) – The highest decomposition level
- **wname** (*str, optional*) – The wavelet type. Default value is `db5`
- **sigma** (*float, optional*) – The damping factor in the Fourier space. Default value is `1.5`
- **axis** (*int, optional*) – The axis index of the theta axis. Default value is `1`.
- **out** (*None or ndarray, optional*) – The output array returned by the function. If it is the same as *arr*, the computation will be done in place. Default value is None, hence a new array is allocated and returned by the function.

Returns **outarr** (*3d array*) – The resulting filtered dataset.

References

neutompy.preproc.preproc.**simple_BHC**(*norm*, *a0=0.0*, *a1=0.0*, *a2=0.02*, *a3=0.0*, *out=None*)

This function performs the simple beam hardening correction (BHC) corresponding to a polynomial correction, given by:

$$s' = s + a_0 s^2 + a_1 s^3 + a_2 s^4 + a_3 s^5$$

where $s = -\ln(I/I_0)$. The user can choose 4 parameters which define the 5-th order polynomial.

Parameters

- **norm** (*3d array*) – Three-dimensional stack of the normalized projections
- **a0** (*float, optional*) – The first term of the polynomial
- **a1** (*float, optional*) – The second term of the polynomial
- **a2** (*float, optional*) – The third term of the polynomial
- **a3** (*float, optional*) – The fourth term of the polynomial

Returns **out** (*3d array*) – Beam hardening correction (BHC) of the input array.

neutompy.preproc.preproc.**zero_clipping_value**(*norm*, *cl=0.01*, *out=None*)

This function clips the values in an array. Values below the threshold value cl are replaced with cl. This function prevents values close to (or below) zero to introduce new artefacts.

$$s' = \max(s, cl)$$

Parameters

- **norm** (*3d array*) – Three-dimensional stack of the normalized projections
- **cl** (*float, optional*) – Clipping value

[3] B. Munch, P. Trtik, F. Marone, M. Stampanoni, Stripe and ring artifact removal with combined wavelet-Fourier filtering, Optics Express 17(10):8567-8591, 2009.

Returns **out** (*3d array*) – Clipping correction of the input array.

3.3.4 neutompy.recon.recon

neutompy.recon.recon.**recon_slice**(*sinogram*, *method*, *pmat*, *parameters=None*, *pixel_size=1.0*, *offset=0*)

This function reconstructs a single sinogram for a 2D parallel beam geaometry.

Parameters

- **sinogram** (*2d array*) – Array representing the sinogram to reconstruct. The rows are projections at different angles.
- **method** (*str*) – A string defining the name of the reconstruction algorithm.

  Available CPU-based methods are: `BP`, `FBP`, `SIRT`, `SART`, `ART`, `CGLS`, `NN-FBP`, `NN-FBP-train`, `MR-FBP`

  Available GPU-based methods are: `BP_CUDA`, `FBP_CUDA`, `SIRT_CUDA`, `SART_CUDA`, `CGLS_CUDA`
- **pmat** (*OpTomo object*) – The ASTRA object that imitates a projection matrix.
- **parameters** (*dict, optional*) – Specific options of the reconstruction algorithm defined in *method*. The complete list of the available options can be found within the ASTRA toolbox documentation: https://www.astra-toolbox.com/docs/algs/index.html.
- **pixel_size** (*float, optional*) – The detector pixel size. If specified in cm the attenuation coefficient values are returned in cm^-1. Default value is 1.0.
- **offset** (*int, optional*) – The offset of the rotation axis with respect to the vertical axis of the detector. If offset is positive the rotation axis is at right-side of the detector vertical axis. If negative, is at left-side. Default value is 0.

Returns **rec** (*2d array*) – The reconstructed slice.

Examples

GPU-based FBP reconstruction with the Hamming filter:

```
>>> import neutomopy as ntp
>>> # read the sinogram
>>> sinogram  = ntp.read_image('file.tiff')
>>> na, nd  = sinogram.shape
>>> angles = np.linspace(0, np.pi*2, na, endpoint=False)
>>> p   = ntp.get_astra_proj_matrix(nd, angles, "FBP_CUDA")
>>> rec = ntp.recon_slice(sinogram, "FBP_CUDA", p, parameters={"FilterType":
↪"hamming"})
```

The filters for FBP_CUDA reconstruction included in the ASTRA toolbox are:

`ram-lak` (default), `shepp-logan`, `cosine`, `hamming`, `hann`, `none`, `tukey`, `lanczos`, `triangular`, `gaussian`, `barlett-hann`, `blackman`, `nuttall`, `blackman-harris`, `blackman-nuttall`, `flat-top`, `kaiser`, `parzen`, `projection`, `sinogram`, `rprojection`, `rsinogram`.

GPU-based SIRT reconstruction with 100 iterations and limiting the pixel values in the range [0,2]:

```
>>> rec =  ntp.recon_slice(sinogram, "SIRT_CUDA", p,
parameters={"iterations":100, "MinConstraint": 0.0, "MaxConstraint" = 2.0 })
```

neutompy.recon.recon.**recon_stack**(*proj*, *method*, *pmat*, *parameters=None*, *pixel_size=1.0*, *offset=0*, *sinogram_order=False*)

This function reconstructs a stack of sinograms or a stack of projections for a 2D parallel beam geaometry.

Parameters

- **proj** (*3d array*) – The data reconstruct. In can be a stack of sinograms or a stack of projections.
- **method** (*str*) – A string defining the name of the reconstruction algorithm.

  Available CPU-based methods are: `BP`, `FBP`, `SIRT`, `SART`, `ART`, `CGLS`, `NN-FBP`, `NN-FBP-train`, `MR-FBP`

  Available GPU-based methods are: `BP_CUDA`, `FBP_CUDA`, `SIRT_CUDA`, `SART_CUDA`, `CGLS_CUDA`
- **pmat** (*OpTomo object*) – The ASTRA object that imitates a projection matrix.
- **parameters** (*dict, optional*) – Specific options of the reconstruction algorithm defined in *method*. The complete list of the available options can be found within the ASTRA toolbox documentation: https://www.astra-toolbox.com/docs/algs/index.html.
- **pixel_size** (*float, optional*) – The detector pixel size. If specified in cm the attenuation coefficient values are returned in cm^-1. Default value is 1.0.
- **offset** (*int, optional*) – The offset of the rotation axis with respect to the vertical axis of the detector. If offset is positive the rotation axis is at right-side of the detector vertical axis. If negative, is at left-side. Default value is 0.
- **sinogram_order** (*bool, optional*) – If `True` the input array is read as a stack of sinograms (0 axis represents the projections y-axis). If `False` the input array is read as a stack of projections (0 axis represents theta). Default value is `False`.

Returns **rec** (*3d array*) – The reconstructed volume.

Examples

GPU-based FBP reconstruction with the Hamming filter:

```
>>> import neutomopy as ntp
>>> # read stack of sinograms
>>> data  = ntp.read_tiff_stack('./sinograms/img_0000.tiff')
>>> _, na, nd  = data.shape
>>> angles = np.linspace(0, np.pi*2, na, endpoint=False)
>>> p   = ntp.get_astra_proj_matrix(nd, angles, "FBP_CUDA")
>>> rec = ntp.recon_stack(data, "FBP_CUDA", p, sinogram_order=True,
                                                    parameters={"FilterType":
↪"hamming"})
```

The filters for FBP_CUDA reconstruction included in the ASTRA toolbox are:

`ram-lak` (default), `shepp-logan`, `cosine`, `hamming`, `hann`, `none`, `tukey`, `lanczos`, `triangular`, `gaussian`, `barlett-hann`, `blackman`, `nuttall`, `blackman-harris`, `blackman-nuttall`, `flat-top`, `kaiser`, `parzen`, `projection`, `sinogram`, `rprojection`, `rsinogram`.

GPU-based SIRT reconstruction with 100 iterations and limiting the values of the reconstructed pixel in the range [0,2]:

```
>>> rec =  ntp.recon_stack(data, "SIRT_CUDA", p, sinogram_order=True,
parameters={"iterations":100, "MinConstraint": 0.0, "MaxConstraint" = 2.0 })
```

neutompy.recon.recon.**reconstruct**(*tomo*, *angles*, *method*, *parameters=None*, *pixel_size=1.0*, *offset=0*, *sinogram_order=False*)

This function reconstructs a dataset of normalized projections or a sinogram for a 2D parallel beam geaometry.

Parameters

- **tomo** (*2d or 3d array*) – It can be a single sinogram, a three-dimensional stack of projections or a three-dimensional stack of sinograms.
- **angles** (*1d array, float*) – The array containing the view angles in radians. For example, for a uniformly spaced scan from 0 to 360 degree with a number of projections nangles: angles = np.linspace(0, 2*np.pi, nangles, endpoint=False)
- **method** (*str*) – A string defining the name of the reconstruction algorithm.

  Available CPU-based methods are: `BP`, `FBP`, `SIRT`, `SART`, `ART`, `CGLS`, `NN-FBP`, `NN-FBP-train`, `MR-FBP`

  Available GPU-based methods are: `BP_CUDA`, `FBP_CUDA`, `SIRT_CUDA`, `SART_CUDA`, `CGLS_CUDA`
- **parameters** (*dict, optional*) – Specific options of the reconstruction algorithm defined in *method*. The complete list of the available options can be found within the ASTRA toolbox documentation: https://www.astra-toolbox.com/docs/algs/index.html.
- **pixel_size** (*float, optional*) – The detector pixel size. If specified in cm the attenuation coefficient values are returned in cm^-1. Default value is 1.0.
- **offset** (*int, optional*) – The offset of the rotation axis with respect to the vertical axis of the detector. If offset is positive the rotation axis is at right-side of the detector vertical axis. If negative, is at left-side. Default value is 0.
- **sinogram_order** (*bool, optional*) – If `True` the input array is read as a stack of sinograms (0 axis represents the projections y-axis). If `False` the input array is read as a stack of projections (0 axis represents theta). Default value is `False`.

Returns **rec** (*2d, 3d array*) – The reconstructed volume if *tomo* is three-dimensional. The reconstructed slice if tomo is a single sinogram.

Examples

GPU-based FBP reconstruction with the Hamming filter:

```
>>> import neutomopy as ntp
>>> # read stack of sinograms
>>> data  = ntp.read_tiff_stack('./sinograms/img_0000.tiff')
>>> _, na, nd  = data.shape
>>> angles = np.linspace(0, np.pi*2, na, endpoint=False)
>>> rec = ntp.reconstruct(data, angles, "FBP_CUDA", sinogram_order=True,
                                                    parameters={"FilterType":
↪"hamming"})
```

The filters for FBP_CUDA reconstruction included in the ASTRA toolbox are:

`ram-lak` (default), `shepp-logan`, `cosine`, `hamming`, `hann`, `none`, `tukey`, `lanczos`, `triangular`, `gaussian`, `barlett-hann`, `blackman`, `nuttall`, `blackman-harris`, `blackman-nuttall`, `flat-top`, `kaiser`, `parzen`, `projection`, `sinogram`, `rprojection`, `rsinogram`.

GPU-based SIRT reconstruction with 100 iterations and limiting the values of the reconstructed pixel in the range [0,2]:

```
>>> rec =  ntp.reconstruct(data, angles, "SIRT_CUDA", sinogram_order=True,
parameters={"iterations":100, "MinConstraint": 0.0, "MaxConstraint" = 2.0 })
```

`neutompy.recon.recon.`**`get_astra_proj_matrix`**(*nd*, *angles*, *method*)

This function returns an object that imitates a projection matrix.

Parameters

- **nd** (*int*) – The number of pixels within a row of the detector.
- **angles** (*1d array, float*) – The array containing the view angles in radians. For example, for a uniformly spaced scan from 0 to 360 degree with a number of projections nangles: angles = np.linspace(0, 2*np.pi, nangles, endpoint=False)
- **method** (*string*) – A string defining the name of the reconstruction algorithm.

  Available CPU-based methods are: `BP`, `FBP`, `SIRT`, `SART`, `ART`, `CGLS`, `NN-FBP`, `NN-FBP-train`, `MR-FBP`

  Available GPU-based methods are: `BP_CUDA`, `FBP_CUDA`, `SIRT_CUDA`, `SART_CUDA`, `CGLS_CUDA`

Returns **pmat** (*OpTomo object*) – The object that imitates a projection matrix.

`neutompy.recon.recon.`**`angles`**(*n_angles*, *start_angle=0.0*, *end_angle=360.0*, *scan_mode='regular'*)

This function returns projection angles in radians for different type of angular scan. It allows to generate uniformly distributed angles in the range [start_angle, end_angle) or a Golden ratio based sequence of projection angles[1].

Parameters

- **n_angles** (*int*) – Number of projections.
- **start_angle** (*float, optional*) – First angle of the sequence in degrees. The parameter is ignored if scan_mode is set to 'golden_ratio'.
- **end_angle** (*float, optional*) – Last angle of the sequence in degrees. The parameter is ignored if scan_mode is set to 'golden_ratio'.
- **scan_mode** (*str, optional*) – It defines the type of the angular scan. Allowed values are: - `regular` uniformly distributed projections in the range [start_angle, end_angle); - `golden_ratio` projections angles according to a Golden ratio based sequence.

Returns **angles** (*1D array*) – The sequence of the projection angles in radians.

[1] T. Kohler, A projection access scheme for iterative reconstruction based on the golden section, IEEE Symposium Conference Record Nuclear Science 2004., Rome, 2004, pp. 3961-3965 Vol. 6.

References

3.3.5 neutompy.recon.optomo

class neutompy.recon.optomo.**OpTomo**(*proj_id*)
Bases: scipy.sparse.linalg.interface.LinearOperator

Object that imitates a projection matrix with a given projector.

This object can do forward projection by using the * operator:

```
W = astra.OpTomo(proj_id)
fp = W*image
bp = W.T*sinogram
```

It can also be used in minimization methods of the scipy.sparse.linalg module:

```
W = astra.OpTomo(proj_id)
output = scipy.sparse.linalg.lsqr(W,sinogram)
```

Parameters **proj_id** (int) – ID to a projector.

BP(*s*, *out=None*)
Perform backprojection.

Output must have the right 2D/3D shape. Input may also be flattened.

Output must also be contiguous and float32. This isn't required for the input, but it is more efficient if it is.

:param : The projection data. :type s: numpy.ndarray :param out: Array to store result in. :type out: numpy.ndarray

FP(*v*, *out=None*)
Perform forward projection.

Output must have the right 2D/3D shape. Input may also be flattened.

Output must also be contiguous and float32. This isn't required for the input, but it is more efficient if it is.

Parameters

- **v** (numpy.ndarray) – Volume to forward project.
- **out** (numpy.ndarray) – Array to store result in.

reconstruct(*method*, *s*, *iterations=1*, *extraOptions=None*)
Reconstruct an object.

Parameters

- **method** (string) – Method to use for reconstruction.
- **s** (numpy.ndarray) – The projection data.
- **iterations** (int) – Number of iterations to use.
- **extraOptions** (dict) – Extra options to use during reconstruction (i.e. for cfg['option']).

rmatvec(*s*)
Implements the transpose operator.

Parameters **s** (numpy.ndarray) – The projection data.

class neutompy.recon.optomo.**OpTomoTransposе**(*parent*)
Bases: scipy.sparse.linalg.interface.LinearOperator

This object provides the transpose operation (`.T`) of the OpTomo object.

Do not use directly, since it can be accessed as member `.T` of an *OpTomo* object.

rmatvec(*v*)
Adjoint matrix-vector multiplication.

Performs the operation y = A^H * x where A is an MxN linear operator and x is a column vector or 1-d array.

Parameters **x** (*{matrix, ndarray}*) – An array with shape (M,) or (M,1).

Returns **y** (*{matrix, ndarray}*) – A matrix or ndarray with shape (N,) or (N,1) depending on the type and shape of the x argument.

Notes

This rmatvec wraps the user-specified rmatvec routine or overridden _rmatvec method to ensure that y has the correct shape and type.

3.3.6 neutompy.postproc.convert

neutompy.postproc.convert.**convert_to_uint**(*arr*, *nbit=16*, *down=None*, *up=None*, *crop_roi=None*)
This function converts a floating point array to 8 or 16 bit unsigned integer array. The resampling can be performed in a user-specified dynamic range. The array can be also cropped by specifying the ROI coordinates.

Parameters

- **arr** (*ndarray*) – The floating point array to convert to unsigned integer array.
- **nbit** (*int*) – The number of bits of the returned array. In must be *8* or *16*, for 8-bit or 16-bit unsigned integer, respectively. Default value is *16*.
- **down** (*float, optional*) – The lower range limit. If *None*, the lower limit is computed as the minimum value of the input array.
- **up** (*float, optional*) – The upper range limit. If *None*, the upper limit is computed as the maximum value of the input array.
- **crop_roi** (*tuple, optional*) – Tuple defining the ROI of the array to crop. E.g. for a 3D stack (smin, smax, rmin, rmax, cmin, cmax), while for a 2D image (rmin, rmax, cmin, cmax). Default value is *None*, which disables the crop of the ROI.

Returns **m** (*ndarray*) – The array resampled to 8-bit or 16-bit unsigned integer.

3.3.7 neutompy.postproc.crop

neutompy.postproc.crop.**get_circular_mask**(*nrow*, *ncol*, *radius=None*, *center=None*)
This function returns a boolean 2D array representing a circular mask. The values outside the circle are *False*, while the inner values are *True*.

Parameters

- **nrow** (*int*) – The number of rows of the mask array.
- **ncol** (*int*) – The number of columns of the mask array.

- **radius** (*float, optional*) – The radius of the circle in pixel. The default value is 0.5 * min(number of rows,number of columns)
- **center** (*tuple of float, optional (yc, xc)*) – A tuple representing the coordinates of the center of the circle, i.e.: (yc, xc). The default value is the center of the input image.

Returns **mask** (*2d array*) – A boolean array that represents the circular mask. The values outside the circle are *False*, while the inner values are *True*.

neutompy.postproc.crop.**circular_crop**(*img*, *axis=0*, *radius=None*, *center=None*, *cval=0*)

This function performs a circular crop of an image. The values outside the circle are replaced with the value cval (default cval=0).

Parameters

- **img** (*2d array*) – The array representing the image to crop.
- **radius** (*float, optional*) – The radius of the circle. The default value is 0.5 * min(number of rows,number of columns)
- **center** (*tuple of float, optional (yc, xc)*) – A tuple representing the coordinates of the center of the circle, i.e.: (yc, xc). The default value is the center of the input image.
- **cval** (*float, optional*) – The value used to fill the region outside the circle.

Returns **out** (*2d array*) – The cropped image.

3.3.8 neutompy.metrics.metrics

neutompy.metrics.metrics.**CNR**(*img*, *croi_signal=[]*, *croi_background=[]*, *froi_signal=[]*, *froi_background=[]*)

This function computes the Contrast-to-Noise Ratio (CNR) as reported in the equation (2.7) of[1]. The ROI of the signal and the background can be defined using two lists of coordinates or two ImageJ .roi files.

Parameters

- **img** (*2d array*) – The array representing the image.
- **croi_signal** (*list*) – List that contains the following coordinate of the signal roi: [rowmin, rowmax, colmin, colmax].
- **croi_background** (*list*) – List that contains the following coordinate of the background roi: [rowmin, rowmax, colmin, colmax].
- **froi_signal** (*string*) – Path of the imagej file containing the rectangular ROI of the signal.
- **froi_background** (*string*) – Path of the imagej file containing the rectangular ROI of the background.

Returns **CNR** (*float*) – The CNR value computed using the ROIs given.

References

neutompy.metrics.metrics.**NRMSE**(*img*, *ref*, *mask='whole'*)

This function computes the Normalized Root Mean Square Error (see eq. 2.9 in[1]) of an image respect to a reference image.

Parameters

[1] D. Micieli et al., A comparative analysis of reconstruction methods applied to Neutron Tomography, Journal of Instrumentation, Volume 13, June 2018.

- **img** (*2d array*) – Test image
- **ref** (*2d array*) – Reference image
- **mask** – It represents the type of the ROI where the NRMSE is computed. You can specify: 'whole' -> full image

  'circ' -> circular mask 'custom' -> custom mask, defined as a boolean matrix of the same shape of the images (img and ref)
- **mask** (*2d bool array*) – Curstom mask of the same shape of the images, where the NRMSE is computed.

Returns **NRMSE** (*float*) – The NRMSE value computed within the roi specified.

neutompy.metrics.metrics.**SSIM**(*img1*, *img2*, *circ_crop=True*, *L=None*, *K1=0.01*, *K2=0.03*, *sigma=1.5*, *local_ssim=False*)

This function computes the Structural Similarity Index (SSIM)[2]. It returns global SSIM value and, optionally, the local SSIM map.

Parameters

- **img1** (*2d array*) – The first image to compare. SSIM index satisfies the condition of simmetry: SSIM(img1, img2) = SSIM(img2, img1)
- **img2** (*2d array*) – The second image to compare. SSIM index satisfies the condition of simmetry: SSIM(img1, img2) = SSIM(img2, img1)
- **circular_crop** (*bool, optional*) – If True (default) the images are cropped with a circular mask, otherwise the SSIM is computed over the entire image.
- **L** (*float, optional*) – The data range of the input images (distance between minimum and maximum possible values). By default, this is estimated from the image data-type.
- **K1** (*float, optional*) – A constant that prevents the division by zero (see[2]).
- **K2** (*float, optional*) – A constant that prevents the division by zero (see[2]).
- **sigma** (*float, optional*) – The standard deviation of the Gaussian filter. This parameter sets the minimum scale at which the quality is evaluated.
- **local_ssim** (*float, optional*) – If True, the function returns the local SSIM map.

Returns

- **ssim** (*float*) – The global SSIM index.
- **map** (*2d array*) – The bidimendional map of the local SSIM index. This is only returned if *local_ssim* is set to True.

References

neutompy.metrics.metrics.**FWHM**(*profile*, *yerr=None*)

This functions computes an edge quality metric from a profile of a sharp edge. The profile is fitted with a generic Gauss sigmoid function. The fitting function was then differentiated and the FWHM of the gaussian obtained is returned by the function. This method is described in detail in[1].

Parameters

- **profile** (*1d array*) – The line profile of the sharp edge.

[2] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing, 13, 600-612. https://ece.uwaterloo.ca/~z70wang/publications/ssim.pdf

- **yerr** (*1d array, optional*) – The vector containing the standard deviation of the edge profile. If not specified the standard deviation of each point is assumed equal to 1.0.

Returns

- **fwhm** (*float*) – The FWHM mean value.
- **fwhm_err** (*float*) – The FWHM error value
- **profile_fitted** (*1d array*) – The Gauss sigmoid function evaluated for the fitting parameters.
- **popt** (*list*) – List containing the fitting parameters.
- **perr** (*list*) – List containing the errors of the fitting parameters.

neutompy.metrics.metrics.**get_line_profile**(*image*, *start=()*, *end=()*, *froi=''*, *ShowPlot=True*, *PlotTitle='Profile'*, *linewidth=1*, *order=1*, *mode='constant'*, *cval=0.0*)

This function returns the intensity profile of an image measured along a line defined by the points:

start = (x_start, y_start) [i.e. (col_start row_start)] end = (x_end, y_en) [i.e. (col_end row_end)]

or an ImageJ .roi file containing the line selection. A plot representing the intensity profile can be shown.

Parameters

- **image** (*ndarray*) – The image grayscale (2D array) or a stack of images (3d array) with shape (slices, rows, columns). Ffor a 3D array the first axis represents the image index.
- **start** (*2-tuple of numeric scalar (float or int) (x y) [i.e. (col row)]*) – The start point of the scan line.
- **end** (*2-tuple of numeric scalar (float or int) (x y) [i.e. (col row)]*) – The end point of the scan line. The destination point is *included* in the profile, in constrast to standard numpy indexing.
- **froi** (*string*) – Path of the imagej file containing the line selection.
- **ShowPlot** (*bool*) – If True a canvas is created representing the Plot Profile.
- **linewidth** (*int, optional*) – Width of the scan, perpendicular to the line
- **order** (*int in {0, 1, 2, 3, 4, 5}, optional*) – The order of the spline interpolation to compute image values at non-integer coordinates. 0 means nearest-neighbor interpolation.
- **mode** (*{'constant', 'nearest', 'reflect', 'mirror', 'wrap'}, optional*) – How to compute any values falling outside of the image.
- **cval** (*float, optional*) – If *mode* is 'constant', what constant value to use outside the image.

Returns **return_value** (*array*) – The intensity profile along the scan line. The length of the profile is the ceil of the computed length of the scan line.

neutompy.metrics.metrics.**GMSD**(*img*, *ref*, *rescale=True*, *map=False*)

This function computes the Gradient Magnitude Similarity Deviation (GMSD). This is a Python version of the Matlab script provided by the authors in[3]

Parameters

- **img** (*2d array*) – Image to compare.

[3] Wufeng Xue, Lei Zhang, Xuanqin Mou, and Alan C. Bovik, "Gradient Magnitude Similarity Deviation: A Highly Efficient Perceptual Image Quality Index", http://www.comp.polyu.edu.hk/~cslzhang/IQA/GMSD/GMSD.htm

- **ref** (*2d array*) – Reference image.
- **rescale** (*bool, optional*) – If True the input images were rescaled in such a way that *ref* has a maximum pixel value of 255. If False no rescaling is performed. Default value is True.
- **map** (*bool, optional*) – If True the GMSD and GMS map are returned in a tuple. Default value is False.

Returns

- **gmsd_val** (*float*) – The GMSD value.
- **gms_map** (*2d array*) – The GMS map, returned only if *map* is True.

References

3.3.9 neutompy.misc.uitools

`neutompy.misc.uitools.`**`get_image_gui`**(*initialdir=''*, *message='Select image...'*)

This function opens a dialog box to select an image (TIFF or FITS) and to get its file path.

Parameters

- **initialdir** (*str, optional*) – String defining the path of the initial folder to open in the dialog box.
- **message** (*str, optional*) – String defining the dialog box title.

Returns **fname** (*str*) – String defining the file path selected using the dialog box.

`neutompy.misc.uitools.`**`save_filename_gui`**(*initialdir=''*, *message='Select folder and the name of the file to save...'*)

This function opens a dialog box to select a file to save and get its file path.

Parameters

- **initialdir** (*str, optional*) – String defining the path of the initial folder to open in the dialog box.
- **message** (*str, optional*) – String defining the dialog box title.

Returns **fname** (*str*) – String defining the file path selected using the dialog box.

`neutompy.misc.uitools.`**`get_folder_gui`**(*initialdir=''*, *message='Select folder...'*)

This function opens a dialog box to select a folder and get its file path.

Parameters

- **initialdir** (*str, optional*) – String defining the path of the initial folder to open in the dialog box.
- **message** (*str, optional*) – String defining the dialog box title.

Returns **fname** (*str*) – String defining the folder path selected using the dialog box.

`neutompy.misc.uitools.`**`get_filename_gui`**(*initialdir=''*, *message='Select file...'*, *ext=None*)

This function opens a dialog box to select a file and get its file path.

Parameters

- **initialdir** (*str, optional*) – String defining the path of the initial folder to open in the dialog box.
- **message** (*str, optional*) – String defining the dialog box title.

- **ext** (*tuple, optional*) – Tuple defining the file types to show. It includes the description and a shell-style wildcards defining the extension of the files. E.g. to filter TIFF images: ((‘Tiff iamges’, ‘*.tiff *.tif’))

Returns **fname** (*str*) – String defining the file path selected using the dialog box.

neutompy.misc.uitools.**get_screen_resolution**()

This function returns the screen resolution as tuple.

Example

```
>>> width, height = ntp.get_screen_resolution()
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

S

W

Z