pyPhase

Max Langer

Dec 01, 2022

CONTENTS

pyPhase is an open-source Python package for phase retrieval from phase contrast images in the Fresnel regime. For an overview, check out the pyPhase manuscript: https://arxiv.org/abs/2012.07942

- Phase retrieval algorithms
- Wave propagation.
- Handling of different image sources and formats
- Tools for pre-processing such as registration of phase contrast images and motion estimation

CHAPTER

ONE

INSTALLATION

Installation is currently through PyPI. Create a virtual environment using your favourite virtual environment manager, verify that pip is installed, then:

```
pip install pyphase
```

PyPhase currently requires Elastix 4.9 installed for registration. To manually install elastix 4.9 go to https://elastix.lumc.nl/download.php

- Unzip the archive.
- Add the path for elastix/bin to your .bashrc: add YOUR_PATH_TO_elastix/bin to your environment variable PATH.
- Add the path for elastix/lib to your .bashrc: add YOUR_PATH_TO_elastix/lib to your environment variable LD_LIBRARY_PATH.

Test your installation:

```
python3 import pyphase
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

2.1 About

pyPhase is an open source phase retrieval package created by *Max Langer*. X-ray propagation-based imaging techniques are well-established at synchrotron radiation and laboratory sources. However, most reconstruction algorithms for such image modalities, also known as phase retrieval algorithms, have been developed specifically for one instrument by and for experts, making the development and spreading of the use of such techniques difficult. Here, we present PyPhase, a free and open-source package for propagation-based near-field phase reconstructions, which is distributed under the CeCILL license. PyPhase implements some of the most popular phase-retrieval algorithms in a highly-modular framework supporting the deployment on large-scale computing facilities. This makes the integration, the development of new phase-retrieval algorithms, and the deployment on different computing infrastructures straight-forward. To demonstrate its capabilities and simplicity, we present its application to data acquired at synchrotron MAX~IV (Lund, Sweden). For more information, read the manuscript describing the package: https://arxiv.org/abs/2012.07942

2.2 Examples

A simple example of usage is given, illustrating the use of the dataset module as well as using directly phase contrast images as input.

```
import pyphase
from pyphase import *
import numpy as np
```

```
#%% Choose image display to use
displayer = utilities.PyplotImageDisplayer()
```

```
#%% Load dataset
data = dataset.NanomaxPreprocessed2D('cpr', version='test')
```

```
#%% Align images using default registrator
# Increase number of resolutions and
pyphase.registrator.parameters.NumberOfResolutions = 8
```

(continues on next page)

(continued from previous page)

```
pyphase.registrator.parameters.MaximumNumberOfIterations = 3000

data.align_projection()
```

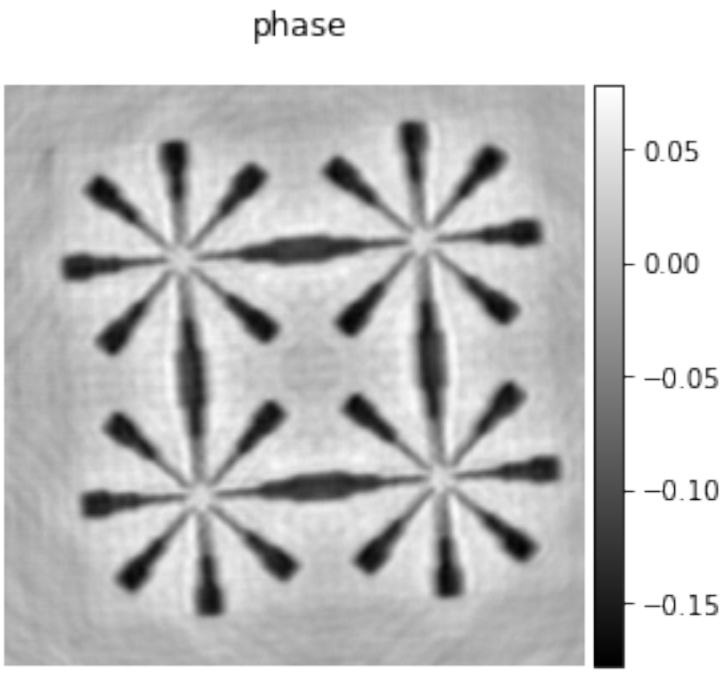
```
Aligning position 1
Found elastix version: 4.900 in '/home/ext-maxlan/elastix/bin/elastix'
Aligning position 2
Aligning position 3
Aligning position 4
```

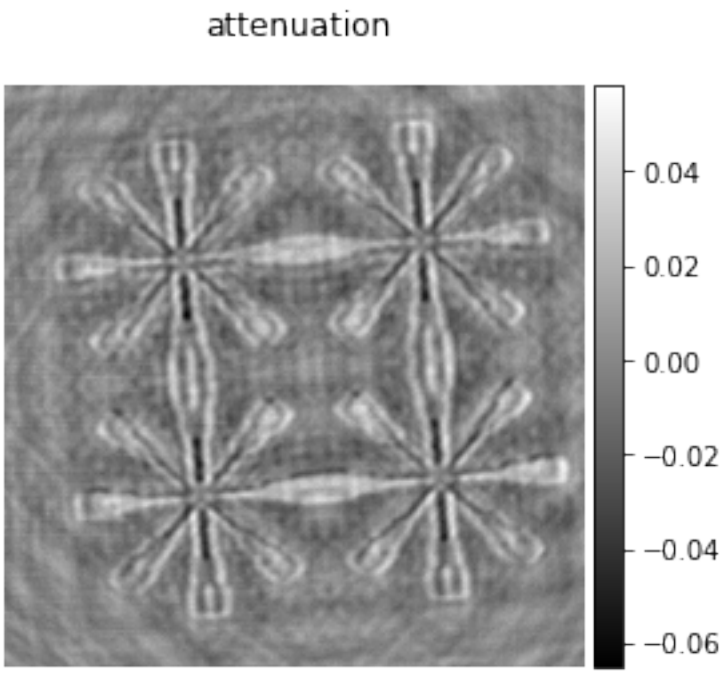
```
#%% Phase retrieval from a dataset using CTF
retriever = phaseretrieval.CTF(data)

# Modify regularisation parameter
retriever.alpha = [1e-3, 1e-8] # [LF, HF]
```

```
#%% Reconstruct
phase, attenuation = retriever.reconstruct_projection(dataset=data, projection=0)
```

```
#%% Display reconstruction
displayer.close_all()
displayer.display(data.get_image(projection=0), title='phase')
displayer.display(data.get_image(projection=0, image_type='attenuation'), title=
↪'attenuation')
```






```
#%% Phase retrieval from images
# Acquisition parameters
energy=13 #keV
effective_distance=np.array([0.010054, 0.0155, 0.0178, 0.019, 0.0203])
pixel_size = np.array([0.005924, 0.006043])*1e-6

nyp = nxp = 4096
ny = nx = 2048
```

```
#%% Load images
ID = np.zeros((len(effective_distance), nyp, nxp))
for N in range(len(effective_distance)):
    ID[N] = data.get_projection(projection=0, position=N, pad=True)
```

```
#%% Phase retrieval from images using HIO_ER
retriever = phaseretrieval.HIO_ER(shape=ID[0].shape, pixel_size=[pixel_size[0], pixel_
↪size[1]], distance=effective_distance, energy=energy, pad=1)

# Modify some parameters
retriever.alpha = [1e-3, 1e-8] # Regularisation parameter used for initialisation
retriever.iterations_hio = 2 # Number of HIO iterations
retriever.iterations_er = 2 # Number of ER iterations
retriever.iterations = 2 # Change number of global iterations
```

```
#%% Reconstruct
phase, attenuation = retriever.reconstruct_image(ID)
```

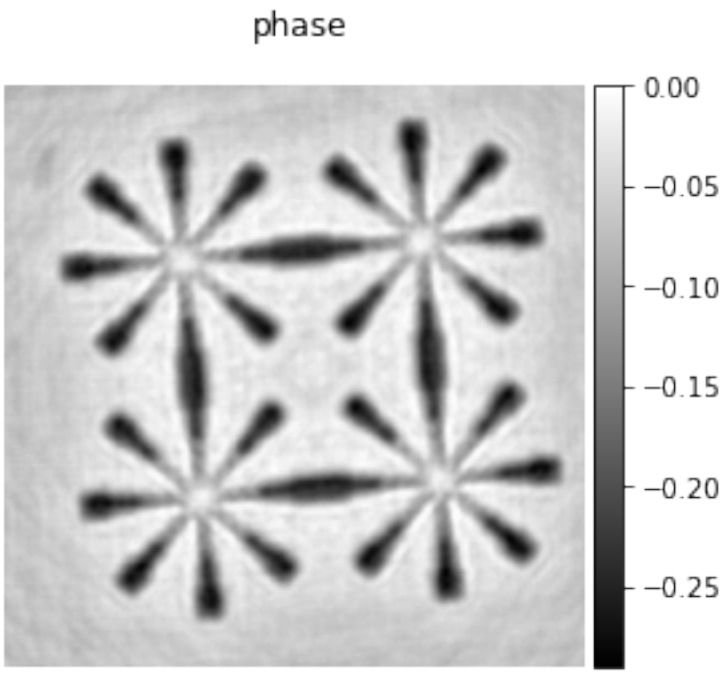
```
========== processing distance 1 ==========
Iteration 0001, error: 1.1e-05
Iteration 0002, error: 7.9e-06
Iteration 0003, error: 6.3e-06
Iteration 0004, error: 5.3e-06
```

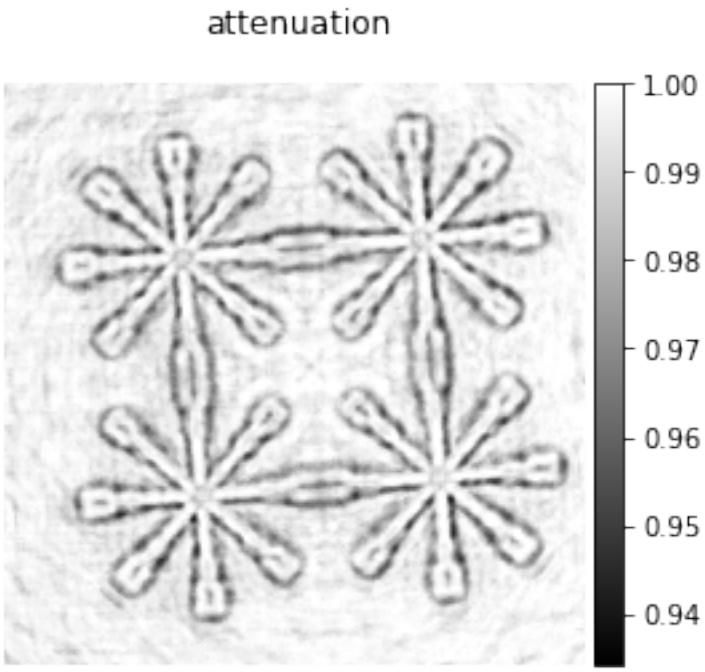
(continues on next page)

(continued from previous page)

```
Iteration 0005, error: 4.7e-06
Iteration 0006, error: 4.2e-06
Iteration 0007, error: 3.8e-06
Iteration 0008, error: 3.5e-06
========== processing distance 2 ==========
Iteration 0001, error: 8.4e-05
Iteration 0002, error: 7e-05
Iteration 0003, error: 6.4e-05
Iteration 0004, error: 6e-05
Iteration 0005, error: 5.7e-05
Iteration 0006, error: 5.4e-05
Iteration 0007, error: 5.2e-05
Iteration 0008, error: 5.1e-05
========== processing distance 3 ==========
Iteration 0001, error: 1.7e-05
Iteration 0002, error: 9.8e-06
Iteration 0003, error: 7e-06
Iteration 0004, error: 5.1e-06
Iteration 0005, error: 4e-06
Iteration 0006, error: 3.3e-06
Iteration 0007, error: 2.8e-06
Iteration 0008, error: 2.4e-06
========== processing distance 4 ==========
Iteration 0001, error: 2.8e-05
Iteration 0002, error: 1.9e-05
Iteration 0003, error: 1.5e-05
Iteration 0004, error: 1.2e-05
Iteration 0005, error: 1.1e-05
Iteration 0006, error: 9.4e-06
Iteration 0007, error: 8.7e-06
Iteration 0008, error: 8e-06
========== processing distance 5 ==========
Iteration 0001, error: 1.8e-05
Iteration 0002, error: 1.2e-05
Iteration 0003, error: 1e-05
Iteration 0004, error: 8.3e-06
Iteration 0005, error: 6.9e-06
Iteration 0006, error: 6e-06
Iteration 0007, error: 5.6e-06
Iteration 0008, error: 5e-06
```

```
#%% Display reconstruction
displayer.close_all()
displayer.display(utilities.resize(phase, [ny, nx]), 'phase')
displayer.display(utilities.resize(attenuation, [ny, nx]), 'attenuation')
```


phase
0.00
−0.05
−0.10
−0.15
−0.20
−0.25


attenuation
1.00
0.99
0.98
0.97
0.96
0.95
0.94

2.3 API reference

This section contains the API reference and usage information for pyPhase.

pyPhase Modules:

2.3.1 pyPhase

pyPhase package

dataset module

class dataset.**Backend**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: object

class dataset.**Dataset**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: object

Abstract class representing a set of recorded images, acquisition parameters, reconstructed images and any intermediary images.

property Lambda
Wavelength in m, calculated from energy.

align_projection(*, *projection=0*, *save_projection=False*)
Aligns images at different positions at one projection angle

Aligns (registers) images taken at different positions using the default registration algorithm (pyphase.registrator).

Parameters

- **projection** (*int, optional*) – Number of projection to register.
- **save_projection** (*bool, optional*) – Save aligned images to file (creates new data files)

align_projections(*, *projections*)
Align a series of projections.

Parameters **projections** (*tuple of ints*) – In the form [start, end]

get_alignment(*, *projection*, *position*)
Get transform parameters for alignment.

Returns **transform_parameters** (*numpy array*) – Array of transform parameters. Size depends on transform used.

property nfx

property nfy

class dataset.**EDF**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.Backend*

property backend_initialized

get_image(*, *projection*, *image_type='phase'*, *Fourier=False*, *pad=True*)
Get reconstructed images (phase by default)

Parameters

- **projection** (*int*) – Number of the projection.
- **image_type** (*str, optional*) – Can be phase or attenuation.
- **Fourier** (*bool*) – Return Fourier transform of image.
- **pad** (*bool*) – Return padded image.

Returns **image** (*ndarray*)

initialize_backend()

read_parameters_backend()
Reads the associated backend parameter file.

write_image(*, *image*, *projection*, *projection_type='phase'*)
Saves images into file.

Parameters

- **data** (*ndarray*) – An ndarray of the image to save.
- **projection_number** (*int*) – The number of the projection to be saved.
- **proj_type** (*str*) – A string containing the prefix for the name of the file to be saved.
- **args** (*int*) – Number of the distance of the projection.

Returns *None* – Saves the images into the file `[prefix]_[projection_number].edf` or `[prefix]_[distance]_[projection_number].edf` into the `name_` directory.

class dataset.**ESRF**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: object

Legacy code for ESRF datasets.

Note: Will be aligned with Nanomax functionality.

GetSinogram(*distance=1*)
Return projections as sinogram

Parameters **distance** (*int*) – Number of the distance of the projection.

Returns *sinogram* – Returns projections stacked as sinogram

align(*interval=100*)
Align(self, RA, interval=100) Align complete data set.

TODO: Should probably check for the last projection also so that it is always included

align_projection(*projection*)
projection = number of projection to register

calculate_axis_position(*distance*)
Calculates the axis position at one position

calculate_axis_positions()
Calculates all axis positions.

calculate_motion()
Estimates motion in the scans based on supplementary images.

calculate_reference_position()
Calculates and sets reference position based on motion

display_alignment(*projection=0*)
Displays alignment and fit.

fit_alignment()
Fits measured alignment parameters with polynomials

get_alignment(*projection*, *distance*)
Returns the transform parameters for alignment at one projection and position

get_image(*, *projection*, *image_type='phase'*, *Fourier=False*, *pad=False*)
Get reconstructed images (phase by default)

Parameters

- **projection** (*int*) – Number of the projection.
- **image_type** (*str, optional*) – Can be phase or attenuation.
- **Fourier** (*bool*) – Return Fourier transform of image.
- **pad** (*bool*) – Return padded image.

Returns **image** (*ndarray*)

get_motion(*projection*, *distance*)
Returns estimation motion at one projection and position.

get_projection(*, *projection*, *position*, *pad=True*, *Fourier=False*, *aligned=True*, *magnification=True*)

property nfx

property nfy

populate()
Tries to find the dataset parameters in the accompanying *.info* and *xml* files. It is called when a parameter file `pyphase_parameter.yml` is not found.

Returns *self* – Returns *self* with the updated attributes.

preprocess()
Runs all preprocessing on the dataset: axis positions, motion, reference position, alignment

write_image(*, *image*, *projection*, *projection_type='phase'*)
Saves images into file.

Parameters

- **data** (*ndarray*) – An ndarray of the image to save.
- **projection_number** (*int*) – The number of the projection to be saved.
- **proj_type** (*str*) – A string containing the prefix for the name of the file to be saved.
- **args** (*int*) – Number of the distance of the projection.

Returns *None* – Saves the images into the file `[prefix]_[projection_number].edf` or `[prefix]_[distance]_[projection_number].edf` into the `name_` directory.

class dataset.**ESRF_dev**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.Dataset*, *dataset.HDF5*

get_projection(*, *projection*, *position*, *pad=True*, *Fourier=False*, *aligned=True*, *magnification=True*)

read_parameters_frontend()

class dataset.**Elettra**(*name: str*, *path: str = '.'*, *version: str = 'aster'*)
Bases: *dataset.Dataset*, *dataset.HDF5*

Class for raw Elettra Synchrotron data sets.

Variables

- **ztot** (*float*) – Focus to detector distance in m.
- **path** (*str*) – Path to dataset.
- **version** (*str*) – Version of Dataset backend structure, e.g. 'TIEHOM'
- **name** (*str*) – Name of dataset
- **projection_prefix** (*str*) – HDF5 path to projection data.
- **reference_position** (*int*) – Position number as reference for alignment (usually highest resolution)
- **darkfield_prefix** (*str*) – HDF5 path prefix to darkfields
- **flatfield_prefix** (*str*) – HDF5 path prefix to flatfields
- **aligned** (*int*) – Flag if dataset is aligned or not.

get_projection(*, *projection*, *position=0*, *pad=True*, *magnification=True*, *aligned=True*, *Fourier=False*)
Read one recorded image.

Parameters

- **projection** (*int*) – Number of projection to read.
- **position** (*int*) – Number of positiion ("distance") to read.
- **pad** (*bool, optional*) – Pads the image.
- **magnification** (*bool, optional*) – Brings the image to the magnification of reference_position.
- **aligned** (*bool, optional*) – Corrects alignment (requires alignment of projections).
- **Fourier** (*bool*) – Returns the Fourier transform of the image.

read_parameters_frontend()

class dataset.**HDF5**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.Backend*

property backend_initialized

get_image(*, *projection*, *image_type='phase'*, *Fourier=False*, *pad=False*)
Get reconstructed images (phase by default)

Parameters

- **projection** (*int*) – Number of the projection.
- **image_type** (*str, optional*) – Can be phase or attenuation.
- **Fourier** (*bool*) – Return Fourier transform of image.
- **pad** (*bool*) – Return padded image.

Returns **image** (*ndarray*)

initialize_backend()

property preprocessing_initialised

read_parameters_backend()

write_dark(*darkfield*, *position*)
Writes a tensor of dark images to file

write_flat(*flatfield*, *position*)
Writes a tensor of flat field images to file

write_image(*, *image*, *projection*, *projection_type='phase'*)
Saves images into file.

Parameters

- **image** (*ndarray*) – An ndarray of the image to save.
- **projection** (*int*) – The number of the projection to be saved.
- **proj_type** (*str*) – A string containing the prefix for the name of the file to be saved.
- **position** (*int*) – Number of the position of the projection (for propagation).

Returns *None* – Saves the images into the file `[prefix]_[projection_number].edf` or `[prefix]_[distance]_[projection_number].edf` into the `name_` directory.

class dataset.**ID16B**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.ESRF_dev*, *dataset.HDF5*

class dataset.**ID19**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.ESRF_dev*, *dataset.HDF5*

read_parameters_frontend()

class dataset.**Nanomax**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.Dataset*, *dataset.HDF5*

Class for raw Nanomax data sets.

Variables

- **ztot** (*float*) – Focus to detector distance in m.
- **path** (*str*) – Path to dataset.
- **version** (*str*) – Version of Dataset backend structure, e.g. 'TIEHOM'
- **name** (*str*) – Name of dataset
- **frames_per_position** (*int*) – Number of frames per position.
- **initial_frame** (*int*) – Number of the initial frame.
- **projection_prefix** (*str*) – HDF5 path to projection data.
- **reference_position** (*int*) – Position number as reference for alignment (usually highest resolution)
- **diode_name** (*str*) – Name of diode (HDF path) for normalisation
- **darkfield_prefix** (*str*) – HDF5 path prefix to darkfields
- **flatfield_prefix** (*str*) – HDF5 path prefix to flatfields
- **cpr_prefix** (*str*) – HDF5 path prefix to flatfield corrected images
- **phase_prefix** (*str*) – HDF5 path prefix to phase maps
- **attenuation_prefix** (*str*) – HDF5 path prefix to attenuation images

- **aligned** (*int*) – Flag if dataset is aligned or not.

get_projection(*, *projection*, *position*, *pad=True*, *magnification=True*, *aligned=True*, *Fourier=False*)
Read one recorded image.

Parameters

- **projection** (*int*) – Number of projection to read.
- **position** (*int*) – Number of positiion ("distance") to read.
- **pad** (*bool, optional*) – Pads the image.
- **magnification** (*bool, optional*) – Brings the image to the magnification of reference_position.
- **aligned** (*bool, optional*) – Corrects alignment (requires alignment of projections).
- **Fourier** (*bool*) – Returns the Fourier transform of the image.

read_parameters_frontend()

class dataset.**NanomaxPreprocessed**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.Dataset*, *dataset.HDF5*

Class for preprocessed Nanomax datasets.

Variables

- **ztot** (*float*) – Focus to detector distance in m.
- **path** (*str*) – Path to dataset.
- **version** (*str*) – Version of Dataset backend structure, e.g. 'TIEHOM'
- **name** (*str*) – Name of dataset
- **correct_alignent** (*int*) – Flag to correct alignment.
- **aligned** (*int*) – Flag for dataset aligned.
- **phase_prefix** (*str*) – HDF5 path prefix to phase maps
- **attenuation_prefix** (*str*) – HDF5 path prefix to attenuation images
- **projection_prefix** (*str*) – HDF5 path to corrected projections
- **reference_position** (*int*) – Position number as reference for alignment (usually highest resolution)
- **energy** (*float*) – Energy in keV
- **ny** (*nx,*) – Number of pixels, horizontal, vertical
- **padding** (*int*) – Padding factor
- **detector_pixel_size** (*float*) – The detector pixel size in µm
- **alpha** (*tuple of floats*) – Regularization parameter in the form [LF, HF]

read_parameters_frontend()

class dataset.**NanomaxPreprocessed2D**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.NanomaxPreprocessed*, *dataset.HDF5*

Class for single projection Nanomax data.

get_projection(*, *projection*, *position*, *pad=True*, *magnification=True*, *aligned=True*, *Fourier=False*)
Read one recorded image.

Parameters

- **projection** (*int*) – Number of projection to read.
- **position** (*int*) – Number of positiion ("distance") to read.
- **pad** (*bool, optional*) – Pads the image.
- **magnification** (*bool, optional*) – Brings the image to the magnification of reference_position.
- **aligned** (*bool, optional*) – Corrects alignment (requires alignment of projections).
- **Fourier** (*bool*) – Returns the Fourier transform of the image.

read_parameters_frontend()

class dataset.**NanomaxPreprocessedTomo**(*name: str*, *path: str = '.'*, *version: str = 'master'*)
Bases: *dataset.NanomaxPreprocessed*, *dataset.HDF5*

Class for preprocessed Nanomax tomographic datasets.

Variables

- **data_basename** (*str*) – File prefix to projection data files.
- **magnification_x,y** (*numpy array*) – Magnification factors for each position, horizontal, vertical
- **pixel_size_x,y** (*numpy array*) – Effective pixel size for each position
- **pixel_size** (*numpy array*) – Effective pixel size at reconstruction position

get_projection(*, *projection*, *position*, *pad=True*, *magnification=True*, *aligned=True*, *Fourier=False*)
Read one recorded image.

Parameters

- **projection** (*int*) – Number of projection to read.
- **position** (*int*) – Number of positiion ("distance") to read.
- **pad** (*bool, optional*) – Pads the image.
- **magnification** (*bool, optional*) – Brings the image to the magnification of reference_position.
- **aligned** (*bool, optional*) – Corrects alignment (requires alignment of projections).
- **Fourier** (*bool*) – Returns the Fourier transform of the image.

read_parameters_frontend()

dataset.**backend**
alias of *dataset.HDF5*

parallelizer module

class parallelizer.**OAR**
Bases: object

Legacy class for parallellisation on OAR. Will be reimplemented as decorator

Launch(*dataset*, *operator*, ***kwargs*)

WriteOarFiles(*DS*)

parallelizer.**SLURM**(*func*)
Decorator for parallellisation on SLURM

parallelizer.**Serial**(*func*)
Decorator for serial processing. Parallelisation decorators work on functions with signature (self, *, dataset, projections)

phaseretrieval module

Functions:

---

class phaseretrieval.**ADMM_CTFhomo**(*dataset=None*, *PSF=[]*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Alternating Direction Method of Multipliers based on CTF (homogeneous) linearization [1]

Parameters

- **ADMM_iterations** (*int*) – Number of ADMM-CTF iterations
- **tau** (*float*) – Penalty parameter of augmented Lagrangian
- **alpha** (*float*) – Penalty parameter of Total Variation (TV) regularization
- **beta_over_delta** (*float*) – Refractive index ratio between beta and delta
- **phys** (*int*) – 0 : No physical constraints 1 : Positivity of attenuation and phase as a constraint

References

[1] Villanueva-Perez Optics Letters 42(6) (2017)

grad(*M*)
Gradient operator.

Parameters **M** (*real np.array*) – Image whose gradient is to be computed.

Returns **gradient** (*real np.array*) – Gradient of image M.

grad_adj(*P*)
Adjoint of gradient operator.

Parameters **P** (*real np.array*) – Image whose divergence is to be computed.

Returns **grad_adj** (*real np.array*) – Adjoint gradient of image P.

inv_block_toeplitz_ctf_betaoverdelta(*b*, *tau*, *beta_over_delta*, *OTF=[]*)
Inverse of CTF operator assuming beta over delta

Parameters

- **u** (*real np.array*) – Input image
- **tau** (*float*) – Penalty parameter of augmented Lagrangian
- **beta_over_delta** (*float*) – Refractive index ratio between beta and delta

Returns **denominator** (*real np.array*) – Inverse of CTF operator

operator_ctf_adjoint(*b*, *beta_over_delta*, *FPSF=[]*)
Compute adjoint of CTF operator assuming beta over delta

Parameters

- **b** (*real np.array*) – Input image
- **beta_over_delta** (*float*) – Refractive index ratio between beta and delta

Returns **numerator** (*real np.array*) – Adjoint of CTF operator

shrinkage(*u*, *kappa*)
Shrinkage operation

Parameters

- **u** (*real np.array*) – Input image.
- **kappa** (*float*)

Returns **u** (*real np.array*) – max(0, |u| - kappa) * sign(u)

class phaseretrieval.**CTF**(*dataset=None*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Contrast Transfer Function [1].

References

[1] Cloetens et al. Appl. Phys. Lett. 75 (1999) 2912

class phaseretrieval.**CTFHOM**(*dataset=None*, *delta_beta=100*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Contrast Transfer Function for homoegeneous objects [1].

References

[1] Villanueva-Perez et al. Optics Letters 42 (2017) 1133

Parameters **delta_beta** (*float, optional*) – Material dependent ratio delta over beta.

class phaseretrieval.**CTFPurePhase**(*dataset=None*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Contrast Transfer Function for pure phase objects [1].

References

[1] Cloetens et al. J. Phys. D: Appl. Phys 29 (1996) 133

class phaseretrieval.**GDTV**(*dataset=None*, *PSF=[]*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Gradient Descent with (smooth) total variation regularization

Parameters

- **iterations** (*int*) – Number of GD iterations
- **epsilon** (*float*) – Smoothing factor of Total Variation (TV) regularization
- **alpha** (*float*) – Penalization for Total Variation (TV) regularization
- **tau** (*float*) – Step size
- **self.omega** (*float*) – Penalization for positivity regularization
- **self.omega_augment_factor** (*float*) – Multiplicative factor for positivity regularization

References

[1]

adjoint_Frechet_derivative(*f*, *P*, *epsilon*)
Adjoint of the Frechet derivative of forward operator.

Parameters

- **f** (*complex np.array*) – f = - iB parametrizes the phase shifts and attenuation B, position the derivative is to be computed
- **P** (*complex np.array*) – Fresnel propagator.
- **epsilon** (*real np.array*) – Input of Adjoint of Frechet derivative

Returns **adjoint_frechet** (*complex np.array*) – Adjoint of the Frechet derivative with respect to f at epsilon

div(*P*)
Divergence operator.

Parameters **P** (*complex np.array*) – Image whose divergence is to be computed.

Returns **divergence** (*complex np.array*) – Divergence of image P.

grad(*M*)
Gradient operator.

Parameters **M** (*complex np.array*) – Image whose gradient is to be computed.

Returns **gradient** (*complex np.array*) – Gradient of image M.

class phaseretrieval.**GaussNewton**(*dataset=None*, *PSF=[]*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Iteratively Regularized Gauss Newton Method [1]

Parameters

- **GaussNewton_iterations** (*int*) – Number of Gauss-Newton iterations

- **ConjugateGradient_iterations** (*float*) – Number of Conjugate Gradient iterations per Gauss-Newton iterations
- **threshold_CG** (*float*) – Threshold for Conjugate Gradient method
- **tau** (*float*) – Step size for Gauss-Newton
- **alpha_reduce_factor** (*float*) – Reduction factor for Tikhonov regularization
- **omega** (*float*) – Penalization for positivity regularization
- **omega_augment_factor** (*float*) – Multiplicative factor for positivity regularization
- **sobolev_exponent** (*float*) – Exponent of Sobolev norm for regularization
- **phys** (*int*) – 0 : No physical constraints 1 : Positivity of attenuation and phase as a regularization

References

[1] Maretzke OSA 24(6) (2016) 6490-6506

Frechet_derivative(*f*, *P*, *epsilon*, *Gx*, *ND*)
Frechet derivative of forward operator.

Parameters

- **f** (*complex np.array*) – f = - iB parametrizes the phase shifts and attenuation B, position the derivative is to be computed
- **P** (*complex np.array*) – Fresnel propagator.
- **epsilon** (*complex np.array*) – Input of Frechet derivative
- **Gx** (*real np.array*) – Gramian matrix of regularization term
- **ND** (*int*) – Number of positions

Returns **frechet** (*real np.array*) – Frechet derivative with respect to f at epsilon

adjoint_Frechet_derivative(*f*, *P*, *epsilon*, *Gx*, *ND*, *support*, *quotient_beta_delta=[]*)
Frechet derivative of forward operator.

Parameters

- **f** (*complex np.array*) – f = - iB parametrizes the phase shifts and attenuation B, position the derivative is to be computed
- **P** (*complex np.array*) – Fresnel propagator.
- **epsilon** (*complex np.array*) – Input of Frechet derivative
- **Gx** (*real np.array*) – Gramian matrix of regularization term
- **ND** (*int*) – Number of positions
- **support** (*np.array*) – Support of object.
- **quotient_beta_delta** (*float*) – Beta over Delta factor if homogeneous object

Returns **adjoint_frechet** (*complex np.array*) – Adjoint of the Frechet derivative with respect to f at epsilon

operator_to_inverse(*f*, *P*, *epsilon*, *alpha_reg*, *gamma*, *Gx*, *ND*, *support*)
(Linear) Operator to inverse with conjugate gradient method

Parameters

- **f** (*complex np.array*) – f = - iB parametrizes the phase shifts and attenuation B, position the derivative is to be computed
- **P** (*complex np.array*) – Fresnel propagator.
- **epsilon** (*complex np.array*) – Input of Frechet derivative
- **alpha_reg** (*float*) – Penalization for Thikonov regularization
- **omega** (*float*) – Penalization for positivity regularization
- **Gx** (*real np.array*) – Gramian matrix of regularization term
- **ND** (*int*) – Number of positions
- **support** (*np.array*) – Support of object.

Returns **adjoint_gradient_inverse** (*complex np.array*) – Operator to inverse with respect to f at epsilon

class phaseretrieval.**GradientDescent**(*dataset=None*, *PSF=[]*, ***kwargs*)

Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Gradient descent algorithm.

Parameters **PSF** (*ndarray with the same shape as images*) – Point spread function (optional)

property alpha

property retriever

class phaseretrieval.**HIO_ER**(*dataset=None*, ***kwargs*)

Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Sequence of Hybrid Input Output [1] and Error Reduction [2].

Variables

- ***retriever*** (PhaseRetrievalAlgorithm2D) – Algorithm for initialisation
- **iterations** (*int*) – Number of global iterations
- **iterations_hio** (*int*) – Number of HIO iterations per iteration
- **iterations_er** (*int*) – Number of ER iterations per iteration
- **step_size_phase** (*float*) – Update step size for the phase
- **step_size_attenuation** (*float*) – Update step size for the attenuation

References

[1] Fienup Appl. Opt. 21 (1982) 2758 [2] Gerchberg & Saxton Optik 35 (1972) 237

property alpha

amplitude_constraint(*wavefront*, *amplitude*, *propagator*, *mask=[]*)

Apply amplitude constraint.

Parameters

- **wavefront** (*complex np.array*) – Wavefront to constrain.
- **amplitude** (*np.array*) – Amplitude to impose.
- **propagator** (*complex np.array*) – Propagator corresponding to effective distance of amplitude.

- **mask** (*np.array, optional*) – Zone to apply constraint.

Returns **wavefront_constrained** (*complex np.array*) – Wavefront after applied constraint.

error_estimate (*wavefront*, *amplitude*, *propagator*, *mask=[]*)
Estimate fit to data.

Parameters

- **wavefront** (*complex np.array*) – Wavefront for estimation.
- **amplitude** (*np.array*) – Amplitude from measured image.
- **propagator** (*complex np.array*) – Fresnel propagator corresponding to effective propagation distance in measured image.
- **mask** (*np.array*) – Restrict estimate to a region of interest.

Returns **error** (*float*) – MSE calculated and measured amplitude.

error_reduction (*wavefront*, *support*)
One iteration of Error Reduction.

Parameters

- **wavefront** (*complex np.array*) – wavefront to update.
- **support** (*np.array*) – Support constraint.

Returns **wavefront_updated** (*complex np.array*) – Updated wavefront.

hybrid_input_output (*wavefront*, *initial_wavefront*, *support*, *step_size_attenuation*, *step_size_phase*)
One iteration of the Hybrid Input Output algorithm.

Parameters

- **wavefront** (*complex np.array*) – Constrained wavefront.
- **initial_wavefront** (*complex np.array*) – Wavefront to update.
- **support** (*np.array*) – Support of object.
- **step_size_attenuation** (*float*) – Step size for attenuation update.
- **step_size_phase** (*float*) – Step size for phase update.

Returns **wavefront_updated** (*complex np.array*) – Updated wavefront.

reconstruct_projection (*dataset*, *projection=0*, *positions=None*, *pad=False*)
Reconstruct one projection from a Dataset object and saves the result.

Parameters

- **dataset** (*Dataset*) – Dataset object to use (not necessarily the same as initialised).
- **projection** (*int, optional*) – Number of projection to reconstruct.
- **positions** (*int or list of ints, optional*) – Subset of positions to use for reconstruction

Returns

- **phase** (*np.array*) – Reconstructed phase.
- **attenuation** (*np.array*) – Reconstructed attenuation.

property retriever

class phaseretrieval.**HPR**(*dataset=None*, ***kwargs*)

Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Hybrid Projection Reflection [1]

Variables

- ***retriever*** (PhaseRetrievalAlgorithm2D) – Algorithm for initialisation
- **iterations** (*int*) – Number of HPR iterations
- **step_size_phase** (*float*) – Update step size for the phase
- **step_size_attenuation** (*float*) – Update step size for the attenuation

References

[1] Bauschke & Combettes OSA 20(6) (2003) 1025-1034

property alpha

amplitude_constraint(*wavefront*, *amplitude*, *propagator*, *mask=[]*)

Apply amplitude constraint.

Parameters

- **wavefront** (*complex np.array*) – Wavefront to constrain.
- **amplitude** (*np.array*) – Amplitude to impose.
- **propagator** (*complex np.array*) – Propagator corresponding to effective distance of amplitude.
- **mask** (*np.array, optional*) – Zone to apply constraint.

Returns **wavefront_constrained** (*complex np.array*) – Wavefront after applied constraint.

error_estimate(*wavefront*, *amplitude*, *propagator*, *mask=[]*)

Estimate fit to data.

Parameters

- **wavefront** (*complex np.array*) – Wavefront for estimation.
- **amplitude** (*np.array*) – Amplitude from measured image.
- **propagator** (*complex np.array*) – Fresnel propagator corresponding to effective propagation distance in measured image.
- **mask** (*np.array*) – Restrict estimate to a region of interest.

Returns **error** (*float*) – MSE calculated and measured amplitude.

hybrid_projection_reflection(*wavefront*, *initial_wavefront*, *support*, *step_size_attenuation*, *step_size_phase*)

One iteration of the Hybrid Projection Reflection.

Parameters

- **wavefront** (*complex np.array*) – Constrained wavefront.
- **initial_wavefront** (*complex np.array*) – Wavefront to update.
- **support** (*np.array*) – Support of object.
- **step_size_attenuation** (*float*) – Step size for attenuation update.

- **step_size_phase** (*float*) – Step size for phase update.

Returns **wavefront_updated** (*complex np.array*) – Updated wavefront.

reconstruct_projection(*dataset*, *projection=0*, *positions=None*, *pad=False*)
Reconstruct one projection from a Dataset object and saves the result.

Parameters

- **dataset** (*Dataset*) – Dataset object to use (not necessarily the same as initialised).
- **projection** (*int, optional*) – Number of projection to reconstruct.
- **positions** (*int or list of ints, optional*) – Subset of positions to use for reconstruction

Returns

- **phase** (*np.array*) – Reconstructed phase.
- **attenuation** (*np.array*) – Reconstructed attenuation.

property retriever

class phaseretrieval.**Mixed**(*dataset*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Mixed approach phase retrieval

Note: Legacy code to be aligned with current APIxs

Lcurve(*dataset*, *projection*)
Calculate the L-curve (for finding regularisation parameter)

create_multimaterial_prior(*data*)
Generate a multi-material prior from a tomographic reconstruction of a contact plane scan

Note: Legacy code to be refactored.

display_Lcurve(*dataset*)
Displays the L-curve

get_prior(*projection*)
Generates a prior estimate of the phase

class phaseretrieval.**NLCG**(*dataset=None*, *PSF=[]*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Non-Linear Conjugate Gradient algorithm.

Parameters **PSF** (*ndarray with the same shape as images*) – Point spread function (optional)

property alpha

property retriever

class phaseretrieval.**NLPDHGM**(*dataset=None*, *PSF=[]*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Exact and Linearised : NonLinear Primal Dual Hybrid Gradient Method [1]

Parameters

- **iterations** (*int*) – Number of NL-PDHGM iterations

- **sigma** (*float*) – Step size for dual variables
- **tau** (*float*) – Step size for primal variables
- **alpha_TGV, beta_TGV** (*float*) – Penalizations for Second order Total Generalized Variation (TGV) for attenuation
- **delta_TV** (*float*) – Penalization for Total Variation (TV) for phase
- **gamma** (*float in [0,1]*) – Over-relaxation parameters for primal variables
- **omega** (*float*) – Penalization for positivity regularization
- **omega_augment_factor** (*float*) – Multiplicative factor for positivity regularization
- **phys** (*int*) – 0 : No physical constraints 1 : Positivity of attenuation and phase as a constraint 2 : Positivity of attenuation and phase as a regularization

References

[1] Valkonen Inverse Problems 30 (2014) 055012

`Frechet_derivative` (*f*, *P*, *epsilon*)
Frechet derivative of forward operator.

Parameters

- **f** (*complex np.array*) – f = - iB parametrizes the phase shifts and attenuation B, position the derivative is to be computed
- **P** (*complex np.array*) – Fresnel propagator.
- **epsilon** (*complex np.array*) – Input of Frechet derivative

Returns **frechet** (*real np.array*) – Frechet derivative with respect to f at epsilon

`adjoint_Frechet_derivative` (*f*, *P*, *epsilon*)
Adjoint of the Frechet derivative of forward operator.

Parameters

- **f** (*complex np.array*) – f = - iB parametrizes the phase shifts and attenuation B, position the derivative is to be computed
- **P** (*complex np.array*) – Fresnel propagator.
- **epsilon** (*real np.array*) – Input of Adjoint of Frechet derivative

Returns **adjoint_frechet** (*complex np.array*) – Adjoint of the Frechet derivative with respect to f at epsilon

`div` (*P*)
Divergence operator.

Parameters **P** (*complex np.array*) – Image whose divergence is to be computed.

Returns **divergence** (*complex np.array*) – Divergence of image P.

`grad` (*M*)
Gradient operator.

Parameters **M** (*complex np.array*) – Image whose gradient is to be computed.

Returns **gradient** (*complex np.array*) – Gradient of image M.

class phaseretrieval.**PDHGM_CTF**(*dataset=None*, *PSF=[]*, ***kwargs*)
Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Chambolle-Pock [1] aka Primal Dual Hybrid Gradient Method (PDHGM) CTF-linearization based

Parameters

- **iterations** (*int*) – Number of PDHGM iterations
- **sigma** (*float*) – Step size for dual variables
- **tau** (*float*) – Step size for primal variables
- **alpha, beta** (*float*) – Penalization for Second order Total Generalized Variation (TGV) for attenuation
- **delta** (*float*) – Penalization for Total Variation (TV) for phase
- **gamma** (*float in [0,1]*) – Over-relaxation parameters for primal variables
- **omega** (*float*) – Penalization for positivity regularization
- **omega_augment_factor** (*float*) – Multiplicative factor for positivity regularization
- **phys** (*int*) – 0 : No physical constraints 1 : Positivity of attenuation and phase as a constraint 2 : Positivity of attenuation and phase as a regularization

References

[1] Chambolle & Pock : Journal of Mathematical Imaging and Vision 40 (2011) 120–145

div(*P*)
Divergence operator.

Parameters **P** (*complex np.array*) – Image whose divergence is to be computed.

Returns **divergence** (*complex np.array*) – Divergence of image P.

grad(*M*)
Gradient operator.

Parameters **M** (*complex np.array*) – Image whose gradient is to be computed.

Returns **gradient** (*complex np.array*) – Gradient of image M.

class phaseretrieval.**PhaseRetrievalAlgorithm2D**(*dataset=None*, ***kwargs*)
Bases: `object`

Base class for 2D phase retrieval algorithms.

Parameters

- **dataset** (*pyphase.Dataset, optional*) – A Dataset type object.
- **shape** (*tuple of ints, optional*) – Size of images (ny, nx) for creation of frequency variables etc.
- **pixel_size** (*float, optional*) – In m.
- **distance** (*list of floats, optional*) – Effective propagation distances in m.
- **energy** (*float, optional*) – Effective energy in keV.
- **alpha** (*tuple of floats, optional*) – Regularisation parameters. First entry for LF, second for HF. Typically [1e-8, 1e-10].
- **pad** (*int*) – Padding factor (default 2).

Variables

- **nx** (*int*) – Number of pixels in horizontal direction.
- **ny** (*int*) – Number of pixels in horizontal direction.
- **pixel_size** (*tuple of floats*) – Pixel size [x, y] in µm.
- **ND** (*int*) – Number of positions.
- **energy** (*float*) – Energy in keV.
- **alpha** (*tuple of floats*) – First entry for LF, second for HF. Tyically [1e-8, 1e-10].
- **distance** (*numpy array*) – Effective propagation distances in m.
- **padding** (*int*) – Padding factor.
- **sample_frequency** (*float*) – Reciprocal of pixel size in lengthscale.
- ***nfx*** (*int*) – Number of samples in Fourier domain (horizontal).
- ***nfy*** (*int*) – Number of samples in Fourier domain (vertical).
- **fx** (*numpy array*) – Frequency variable (horizontal), calculated by frequency_variable.
- **fy** (*numpy array*) – Freuency variable (vertical).
- **alpha_cutoff** (*float*) – Cutoff frequency for regularization parameter in normalized frequency.
- **alpha_slope** (*float*) – Slope in regularization parameter.

Notes

Takes either a Dataset type object with keyword dataset (which contains all necessary parameters), or parameters as above.

property Alpha
Image implementation of regularisation parameter (np.array)

property Fresnel_number
Fresnel number at each position, calculated from energy and distance (float)

property Lambda
Wavelength based on energy (float)

frequency_variable(*nfx*, *nfy*, *sample_frequency*)
Calculate frequency variables.

Parameters

- **nfx** (*int*) – Number of samples in x direction
- **nfy** (*int*) – Number of samples in y direction
- **sample_frequency** (*float*) – Reciprocal of pixel size in 1/m

Returns *nparray* – Frequency variables as an array of size [nfy, nfx, 2]

Notes

Follows numpy FFT convention. Zero frequency at [0,0], [1:n//2] contain the positive frequencies, [n//2 + 1:] n the negative frequencies in increasing order starting from the most negative frequency.

property nfx

property nfy

reconstruct_image(*image*, *positions=None*, *pad=False*)
Template for reconstructing an image given as argument.

Parameters

- **image** (*numpy.array*) – A phase contrast image or an ndarray of images stacked along the first dimension.
- **positions** (*int or list of ints, optional*) – Subset of positions to use for reconstruction.

Note: Calls _algorithm (container purely for algorithm part).

Returns

- **phase** (*numpy.array*)
- **attenuation** (*numpy.array*)

reconstruct_projection(*dataset*, *projection=0*, *positions=None*, *pad=True*)
Reconstruct one projection from a Dataset object and saves the result.

Parameters

- **dataset** (*Dataset*) – Dataset object to use (not necessarily the same as initialised).
- **projection** (*int, optional*) – Number of projection to reconstruct.
- **positions** (*int or list of ints, optional*) – Subset of positions to use for reconstruction

Returns

- **phase** (*np.array*) – Reconstructed phase.
- **attenuation** (*np.array*) – Reconstructed attenuation.

reconstruct_projections(*, *dataset*, *projections*)
Reconstruct a range of projections (parallelized function).

Parameters

- **dataset** (*pyphase.Dataset*) – Dataset to reconstruct.
- **projections** (*list of int*) – In the form [start, end]

simple_propagator(*pxs*, *Lambda*, *z*)
Creates a Fresnel propagator.

Parameters

- **pxs** (*float*) – Pixel size in µm.
- **Lambda** (*float*) – Wavelength in m.
- **z** (*float*) – Effective propagation distance in m.

Returns **H** (*nparray*) – Fresnel propagator.

Notes

Temporary implementation by Y. Zhang. Will be integrated with the propagator module.

class phaseretrieval.**RAAR**(*dataset=None*, ***kwargs*)

Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Relaxed averaged alternating reflections [1]

Variables

- ***retriever*** (PhaseRetrievalAlgorithm2D) – Algorithm for initialisation
- **iterations** (*int*) – Number of RAAR iterations
- **step_size_phase** (*float*) – Update step size for the phase
- **step_size_attenuation** (*float*) – Update step size for the attenuation

References

[1] Luke Inverse Problems 21 (2005) 3750

property alpha

amplitude_constraint(*wavefront*, *amplitude*, *propagator*, *mask=[]*)

Apply amplitude constraint.

Parameters

- **wavefront** (*complex np.array*) – Wavefront to constrain.
- **amplitude** (*np.array*) – Amplitude to impose.
- **propagator** (*complex np.array*) – Propagator corresponding to effective distance of amplitude.
- **mask** (*np.array, optional*) – Zone to apply constraint.

Returns **wavefront_constrained** (*complex np.array*) – Wavefront after applied constraint.

error_estimate(*wavefront*, *amplitude*, *propagator*, *mask=[]*)

Estimate fit to data.

Parameters

- **wavefront** (*complex np.array*) – Wavefront for estimation.
- **amplitude** (*np.array*) – Amplitude from measured image.
- **propagator** (*complex np.array*) – Fresnel propagator corresponding to effective propagation distance in measured image.
- **mask** (*np.array*) – Restrict estimate to a region of interest.

Returns **error** (*float*) – MSE calculated and measured amplitude.

reconstruct_projection(*dataset*, *projection=0*, *positions=None*, *pad=False*)

Reconstruct one projection from a Dataset object and saves the result.

Parameters

- **dataset** (*Dataset*) – Dataset object to use (not necessarily the same as initialised).
- **projection** (*int, optional*) – Number of projection to reconstruct.
- **positions** (*int or list of ints, optional*) – Subset of positions to use for reconstruction

Returns

- **phase** (*np.array*) – Reconstructed phase.
- **attenuation** (*np.array*) – Reconstructed attenuation.

relaxed_averaged_alternating_reflections(*wavefront*, *initial_wavefront*, *support*, *step_size_attenuation*, *step_size_phase*)

One iteration of the Relaxed Averaged Alternating Reflections algorithm.

Parameters

- **wavefront** (*complex np.array*) – Constrained wavefront.
- **initial_wavefront** (*complex np.array*) – Wavefront to update.
- **support** (*np.array*) – Support of object.
- **step_size_attenuation** (*float*) – Step size for attenuation update.
- **step_size_phase** (*float*) – Step size for phase update.

Returns **wavefront_updated** (*complex np.array*) – Updated wavefront.

property retriever

class phaseretrieval.**TIEHOM**(*dataset=None*, *delta_beta=500*, ***kwargs*)

Bases: *phaseretrieval.PhaseRetrievalAlgorithm2D*

Transport of Intensity Equation for homogeneous objects (or "Paganin's algorithm") [1]

Parameters **delta_beta** (*float, optional*) – Material dependent ratio delta over beta.

References

[1] Paganin et al. J. Microsc. 206 (2002) 33

property delta_beta

Material dependent ratio delta over beta (float).

propagator module

class propagator.**CTF**(***, *dataset=None*, *shape=None*, *energy=None*, *pixel_size=None*, *distance=None*, *pad=2*, *oversampling=4*)

Bases: *propagator.Propagator*

Propagates using the CTF. Legacy code to be aligned with Fresnel.

PropagateProjection(*dataset*, *projection*, *distance*)

class propagator.**Fresnel**(***, *dataset=None*, *shape=None*, *energy=None*, *pixel_size=None*, *distance=None*, *pad=2*, *oversampling=4*)

Bases: *propagator.Propagator*

Propagator using Fresnel transform

class propagator.**Propagator**(***, *dataset=None*, *shape=None*, *energy=None*, *pixel_size=None*, *distance=None*, *pad=2*, *oversampling=4*)

Bases: object

propagate_image(*amplitude*, *phase*, *position_number=None*, *oversampled=False*)

Propagate a wavefront created from two images.

propagate_projection(*dataset*, *position_number=None*, *projection=None*, *oversampled=False*)
Propagate one projection.

Parameters

- **dataset** (*pyphase.Dataset, optional*) – Datset with projection data.
- **position_number** (*int, optional*) – Which position to propagate to
- **projection** (*int, optional*) – Which projection to propagate
- **phase** (*ndarray, optional*) – Phase of wave to propagate
- **attenuation** (*ndarray, optional*) – Amplitude of wave to propagate
- **position** (*float*) – Effective propagation distance
- **oversampled** (*bool*) – True if imput images are already oversampled

pyphase module

tomography module

class tomography.**PyHST**
Bases: object

Tomographic operations with PyHST

ForwardProject(*DS*, *volume='phase'*)
Generate projections from a reconstructed dataset

reconstruct(*DS*, *volume='phase'*)
Tomographic reconstruction

class tomography.**Tomography**
Bases: object

Class for tomographic operations, for use with 3D iterative methods

Note: Legacy code to be aligned with current API

utilities module

class utilities.**ElastixAffine**
Bases: *utilities.RegistrationAlgorithm*

Affine registration algorithm using Elastix.

Variables

- **parameters** (*Parameters*) – Elastix standard parameters
- **number_of_parameters** (*int, default=6*) – Number of parameters in the transform

class utilities.**ElastixRigid**
Bases: *utilities.RegistrationAlgorithm*

Rigid registration algorithm using Elastix.

Variables

- **parameters** (*Parameters*) – Elastix standard parameters
- **number_of_parameters** (*int, default=3*) – Number of parameters in the transform

class utilities.**ElastixSimilar**
Bases: *utilities.RegistrationAlgorithm*

Similarity transform registration algorithm using Elastix

Variables

- **parameters** (*Parameters*) – Elastix standard parameters
- **number_of_parameters** (*int, default=4*) – Number of parameters in the transform

class utilities.**ImageDisplayer**
Bases: object

Wrapper class for image display.

class utilities.**PyplotImageDisplayer**
Bases: *utilities.ImageDisplayer*

Interface to Pyplot for image display.

Notes

With the idea to make the choice of display package flexible.

close_all()

display(*image*, *title=''*, *vmin=None*, *vmax=None*)
Display an image

Parameters

- **image** (*nparray*) – The image to be displayed.
- **title** (*str, optional*) – Title of figure.
- **vmin** (*optional*) – Lower limit of contrast range.
- **vmax** (*optional*) – Upper limit of contrast range.

display_stack(*stack*)

class utilities.**RegistrationAlgorithm**
Bases: object

Abstract class for registration algorithms

apply_transformation(*image*, *transform_parameters*, ***kwargs*)
Apply an image transform from image registration.

Parameters

- **image** (*ndarray*) – The image to transform.
- **transform_parameters** (*array*) – Parameters of the transform (length depends on registration algorithm used (number_of_parameters))

Returns *transformed_image* – The transformed image.

register(*moving_image*, *stationary_image*)
Register moving_image to stationary_image.

Parameters

- **moving_image** (*ndarray*) – The image to register.
- **stationary_image** (*ndarray*) – The image to register to.

Returns

- **field** (*ndarray*) – The calculated deformation field.
- **transformed_moving_image** (*ndarray*) – The deformed moving image.
- **transform_parameters** (*array*) – The calculated transform parameters. Lenght varies with the number of parameters in the chosen algorithm (number_of_parameters)

class utilities.**StackViewer**(*X*, *ax*)
Bases: object

Functionality to browse stacks.

utilities.**resize**(*image*, *shape*, *pad_type='edge'*)
Resizes an image by either cutting out the centre or padding.

Assumes images are stored along fist dimension.

utilities.**update**(*title*, *position*, *target*)

PYTHON MODULE INDEX

INDEX

F

G

H

I

L

M

N

O

P

R

S

T

U

W