PyLorentz

Release 1.0.0

Jun 27, 2023

Contents:

PyLorentz is a codebase designed for analyzing Lorentz Transmission Electron Microscopy (LTEM) data. There are three primary features and functions:

- PyTIE – Reconstructing the magnetic induction from LTEM images using the Transport of Intensity Equation (TIE)
- SimLTEM – Simulating phase shift and LTEM images from a given magnetization
- GUI – GUI alternative to Jupyter Notebook for PyTIE

CHAPTER 1

PyLorentz Features and Code

The PyLorentz code is written to be accessible and easily integrated into existing work-flows. However it is still possible to get wrong results if one doesn't have familiarity with the systems involved, which in this case are both the materials involved as well as LTEM. It can be easy to arrive at results that aren't physically possible, especially when simulating samples. PyLorentz can not protect against this beyond providing a few *references* that give some background on LTEM.

Additionally, while the TIE GUI is a complete product intending to make PyLorentz accessible to everyone, to access all of the features and get the most out of this product a user would find it helpful to be familiar with basic python.

1.1 Getting started

With the exception of the gui, the PyLorentz code is intended to be run in Jupyter notebooks and several examples are provided. You can clone the repo from the github page, fork the project, or download the files directly in a .zip.

Several standard packages are required which can be installed with conda or pip, and .yml files are included in the `/envs/` folder. Select the appropriate file for your system and create the environment from a command line with:

```
>>> conda env create -f environment.yml
```

Activate with either:

```
>>> source activate PyLorentz
```

or:

```
>>> conda activate PyLorentz
```

depending on operating system before opening a notebook.

1.1.1 Example Data

We recommend running the template files provided in the `/Examples/` directory with the provided example data. The PyTIE template will show you how to download an experimental dataset from the Materials Data Facility, which contains a through focus series (tfs) in both flipped and unflipped orientations as well as an aligned image stack. These files are ready to be used in the `TIE_template.ipynb`.

For `SIM_template.ipynb`, there is an `example_mumax.ovf` file already in the repository, which can be used directly.

1.1.2 Data Organization

PyTIE requires all of the image data to be contained within one main working directory. The exact setup will depend on if you have one single through focus series (tfs) or two.

If you have both a flip and unflip stack your data should be set up as follows:

```
datafolder/    flip/      -im1.dm3
                          -im2.dm3
                             ...
                          +im1.dm3
                          +im2.dm3
                             ...
                          0im.dm3

               unflip/    -im1.dm3
                          -im2.dm3
                              .
                              .
                          +im1.dm3
                          +im2.dm3
                              .
                              .
                           0im.dm3

               flsfile.fls
               full_align.tif
```

If your flip and unflip filenames aren't the same you can also have two fls files, one for each tfs. In that case just change the argument in load_data: `flip_fls_file = "flip/fls/path"`

However if you have just one stack (no flip stack) then your data should be in a folder labeled 'tfs'

```
datafolder/    tfs/       -im1.dm3
                          -im2.dm3
                             ...
                          +im1.dm3
                          +im2.dm3
                             ...
                           0im.dm3
               flsfile.fls
               full_align.tif
```

The `full_align.tif` is a 3D tifstack of the aligned flipped and unflipped images. It is ordered from most underfocus to most overfocus, first unflipped then flipped images, e.g. for a 5 image series:

```
[-2 , -1 , 0 , +1 , +2 , -2f , -1f , 0f , +1f , +2f]
```

1.1.3 The .fls file

The fls file is a text file that makes it easier for the program to read a tfs of images. It contains, in order:

- The number of images in the through focal series
- Names of the images from most underfocus to most overfocus; the middle image should be the in-focus image.
- The defocus values, which should be the same for both over and under focus. There should be $\frac{\text{images}-1}{2}$ total defocus values.

For example, the fls file would be the following for a 5-image stack:

```
5
under2.dm3
under1.dm3
infocus.dm3
over1.dm3
over2.dm3
defocus_val_1
defocus_val_2
```

If you have only one tfs, then you will only have one fls file. If you have two tfs and the files are named differently, you will need two fls files. If the files are named symmetrically between the flip and unflip images, only one fls file is needed (as is the case in the example data).

1.2 API

The API is divided roughly into two sections. All of the code required for TIE reconstructions and displays is contained with `PyTIE`, though much is also used in `SimLTEM` which contains the LTEM simulation code.

1.2.1 PyTIE

The following modules contain the base code for the TIE reconstruction, as well as display functions and the microscope class which is used for simulating LTEM images.

- `TIE_reconstruct` contains the final setup and TIE solver
- `TIE_params` is a class that contains the data and reconstruction parameters.
- `microscopes` is a class containing all microscope parameters as well as methods for simulating LTEM images.
- `TIE_helper` has loading and display functions
- `colorwheel` is for visualizing 2D vector images.
- `longitudinal_deriv` still has bugs, but calculates the intensity derivative through a full stack rather than through the central difference method.

The TIE reconstruction for the magnetic component of the electron phase shift has been shown to be quantitative when verified against simulated images. This is not the case for the electrostatic component, however. While the returned `phase_e` image will be qualitatively correct it is likely to be scaled incorrectly.

TIE_reconstruct module

TIE_params module

Class for TIE reconstruction parameters.

A class for holding TIE images and reconstruction parameters. Also has several methods relating to doing the reconstruction, namely making masks and interactively cropping the image stacks.

AUTHOR: Arthur McCray, ANL, Summer 2019

class TIE_params.**TIE_params**(*imstack=None*, *flipstack=[]*, *defvals=None*, *scale=None*, *flip=None*, *data_loc=None*, *no_mask=False*, *v=1*)

Bases: object

An object for holding the data and parameters for the reconstruction.

Some params will be obtained directly from the metadata, others given by the user. Also holds some other useful values. An instance is created by the load_data() function in TIE_helper.py.

For more information on how to organize the directory and load the data, as well as how to setup the .fls file please refer to the README or the TIE_template.ipynb notebook.

imstack
A list of numpy arrays, one per image in the through focus series (tfs).

Type list

flipstack
A list of numpy arrays for the flip tfs if there is one.

Type list

defvals
List of defocus values in nm from least to most defocus. This assumes a symmetric defocus over/under, so expects 2 values for a 5 image tfs.

Type list

flip
Boolean for whether or not to reconstruct using the flip stack. Even if the flip stack exists the reconstruction will only use the unflip stack if self.flip = False.

Type bool

data_loc
String for location of data folder.

Type str

no_mask
Eliminates mask (generally used with simulated data).

Type bool

num_files
Equiv to len(self.imstack)

Type int

shape
Shape of original image data (y, x)

Type tuple

scale
Scale of images (nm/pixel). Taken from the dm3/tif metadata or set manually.

Type float

rotation
The rotation to apply to the image before reconstruction in deg.

Type float, int

x_transl
The x_translation to apply to the image before reconstruction in pix.

Type int

y_transl
The y_translation to apply to the image before reconstruction in pix.

Type int

infocus
Averaged infocus image between unflip and flip stacks. If no flip stack, just unflip infocus image.

Type 2D array

qi
2D inverse frequency array, possibly modified with Tikhonov filter.

Type 2D array

crop
Region of interest in pixels used to select the are to be reconstructed. Initialized to full image.

Type dict

mask
Binary mask made form all the images. 1 where all images have nonzero data, 0 where any do not. Made by self.make_mask()

Type 2D array

make_mask(*imstack=None*, *threshold=0*)
Sets self.mask to be a binary bounding mask from imstack and flipstack.

Makes all images binary using a threshold value, and then multiplies these arrays. Can also take a stack of images that aren't signal2D.

The inverse Laplacian reconstruction does not deal well with a mask that is all ones, that is accounted for in TIE() function rather than here.

Parameters

- **imstack** (*list*) – (*optional*) List of arrays or Signal2D objects from which to make mask Default will use self.imstack and self.flipstack
- **threshold** (*float*) – (*optional*) Pixel value with which to threshold the images. Default is 0.

Returns None. Assigns result to self.mask()

pre_Lap(*pscope*, *def_step=1*)
Scaling prefactor used in the TIE reconstruction.

Parameters

- **pscope** (Microscope object) – Microscope object from microscopes.py

- **def_step** (*float*) – The defocus value for which is being reconstructed. If using a longitudinal derivative, def_step should be 1.

Returns Numerical prefactor

Return type float

reset_crop()
Reset the ptie.crop() region to the full image.

select_ROI(*infocus=True*)
Select a rectangular region of interest (ROI) and assign it to ptie.crop

Parameters **infocus** (*bool, optional*) – Whether to select a region from the infocus image. For some datasets the infocus image will have no contrast, and therefore set infocus=False to select a region from the most underfocused image. Defaults to True.

TIE_params.**get_dist**(*pos1*, *pos2*)
Distance between two 2D points

Parameters

- **pos1** (*list*) – [y1, x1] point 1
- **pos2** (*list*) – [y2, x2] point 2

Returns Euclidean distance between the two points

Return type float

microscopes module

A class for microscope objects.

Author: CD Phatak, ANL, 20.Feb.2015.

class microscopes.**Microscope**(*E=200000.0*, *Cs=1000000.0*, *Cc=5000000.0*, *theta_c=0.0006*, *Ca=0.0*, *phi_a=0*, *def_spr=120.0*, *verbose=False*)
Bases: object

Class for Microscope objects.

A class that describes a microscope for image simulation and reconstruction purposes. Along with accelerating voltage, aberrations, and other parameters, this also contains methods for propagating the electron wave and simulating images.

Notes:

- When initializing a Microscope you can set verbose=True to get a printout of the parameters.
- Unlike in TIE_reconstruct, here the qq frequency spectrum is expected to be shifted, i.e. rather than four dark corners it's a dark circle.

E
Accelerating voltage (V). Default 200kV.

Type float

Cs
Spherical aberration (nm). Default 1mm.

Type float

Cc
Chromatic aberration (nm). Default 5mm.

Type float

theta_c
Beam coherence (rad). Default 0.6mrad.

Type float

Ca
2-fold astigmatism (nm). Default 0.

Type float

phi_a
2-fold astigmatism angle (rad). Default 0.

Type float

def_spr
Defocus spread (nm). Default 120nm.

Type float

defocus
Defocus of the microscope (nm). Default 0nm.

Type float

lam
Electron wavelength (nm) calculated from E. Default 2.51pm.

Type float

gamma
Relativistic factor (unitless) from E. Default 1.39.

Type float

sigma
Interaction constant (1/(V*nm)) from E. Default 0.00729.

Type float

BackPropagateWave(*ImgWave*, *qq*, *del_px*)
Back-propagate an image wave to get the object wave.

This function will back-propagate the image wave function to the object wave plane by convolving with exp(+i*Chiq). The damping envelope is not used for back propagation. Returns ObjWave in real space.

Parameters

- **ObjWave** (*2D array*) – Object wave function.
- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Realspace object wave function. Complex 2D array same size as ObjWave.

Return type ndarray

PropagateWave(*ObjWave*, *qq*, *del_px*)
Propagate object wave function to image plane.

This function will propagate the object wave function to the image plane by convolving with the transfer function of microscope, and returns the complex real-space ImgWave

Parameters

- **ObjWave** (*2D array*) – Object wave function.
- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Realspace image wave function. Complex 2D array same size as ObjWave.

Return type ndarray

getBFPImage(*ObjWave*, *qq*, *del_px*)
Produce the image in the backfocal plane (diffraction)

Parameters

- **ObjWave** (*2D array*) – Object wave function.
- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Realspace image wave function. Real-valued 2D array same size as ObjWave.

Return type ndarray

getChiQ(*qq*, *del_px*)
Calculate the phase transfer function.

Parameters

- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns 2D array same size as qq.

Return type ndarray

getDampEnv(*qq*, *del_px*)
Calculate the complete damping envelope: spatial + temporal

Parameters

- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Damping envelope. 2D array same size as qq.

Return type ndarray

getImage(*ObjWave*, *qq*, *del_px*)
Produce the image at the set defocus using the methods in this class.

Parameters

- **ObjWave** (*2D array*) – Object wave function.
- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Realspace image wave function. Real-valued 2D array same size as ObjWave.

Return type ndarray

getOptDef(*qq*, *del_px*)
Calculate the Optimum or Lichte defocus (for holography).

Parameters

- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Optimum defocus (nm)

Return type float

getSchDef()
Calculate the Scherzer defocus

getTransferFunction(*qq*, *del_px*)
Generate the full transfer function in reciprocal space

Parameters

- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)

Returns Transfer function. 2D array same size as qq.

Return type ndarray

setAperture(*qq*, *del_px*, *sz*)
Set the objective aperture

Parameters

- **qq** (*2D array*) – Frequency array
- **del_px** (*float*) – Scale (nm/pixel)
- **sz** (*float*) – Aperture size (nm).

Returns Sets self.aperture.

Return type None

TIE_helper module

colorwheel module

Creates RGB images from vector fields.

This module contains several routines for plotting colormaps from input data consisting of 2D images of the vector field. The output image will be stored as a tiff color image. There are options to save it using a custom RGB or standard HSV color-wheel.

Author: Arthur McCray, C. Phatak, ANL, Summer 2019.

colorwheel.**color_im**(*Bx*, *By*, *Bz=None*, *rad=None*, *hsvwheel=True*, *background='black'*)
Make the RGB image from x and y component vector maps.

The color intensity corresponds to the in-plane vector component. If a z-component is given, it will map from black (negative) to white (positive).

Parameters

- **Bx** (*2D array*) – (M x N) array consisting of the x-component of the vector field.
- **By** (*2D array*) – (M x N) array consisting of the y-component of the vector field.
- **Bz** (*2D array*) – optional (M x N) array consisting of the y-component of the vector field.

- **rad** (*int*) – (*optional*) Radius of color-wheel in pixels. (default None -> height/16) Set rad = 0 to remove color-wheel.
- **hsvwheel** (*bool*) –
  - True – (default) use a standard HSV color-wheel (3-fold)
  - False – use a four-fold color-wheel
- **background** (*str*) –
  - 'black' – (default) magnetization magnitude corresponds to value.
  - 'white' – magnetization magnitude corresponds to saturation.

Returns Numpy array (M x N x 3) containing the color-image.

Return type ndarray

colorwheel.**colorwheel_HSV**(*rad*, *background*, *z=False*, ***kwargs*)

Creates an HSV color-wheel as a np array to be inserted into the color-image.

Parameters

- **rad** (*int*) – (*optional*) Radius of color-wheel in pixels. (default None -> height/16) Set rad = 0 to remove color-wheel.
- **background** (*str*) –
  - 'black' – (default) magnetization magnitude corresponds to value.
  - 'white' – magnetization magnitude corresponds to saturation.

Returns Numpy array of shape (2*rad, 2*rad).

Return type ndarray

colorwheel.**colorwheel_RGB**(*rad*)

Makes a 4-quadrant RGB color-wheel as a np array to be inserted into the color-image

Parameters **rad** (*int*) – (*optional*) Radius of color-wheel in pixels. (default None -> height/16) Set rad = 0 to remove color-wheel.

Returns Numpy array of shape (2*rad, 2*rad).

Return type ndarray

colorwheel.**dist**(*ny*, *nx*, *shift=False*)

Creates a frequency array for Fourier processing.

Parameters

- **ny** (*int*) – Height of array
- **nx** (*int*) – Width of array
- **shift** (*bool*) – Whether to center the frequency spectrum.
  - False: (default) smallest values are at the corners.
  - True: smallest values at center of array.

Returns Numpy array of shape (ny, nx).

Return type ndarray

longitudinal_deriv module

Generate longitudinal derivatives through stack of intensity values.

This file contains routines for generating the longitudinal derivative through a stack of images by fitting a quadratic polynomial to the intensity values for each (y,x) pixel.

Author: Arthur McCray, ANL, May 2020.

`longitudinal_deriv.`**`polyfit_deriv`**(*stack*, *defvals*, *v=1*)

Calculates a longitudinal derivative of intensity values taken at different defocus values. Expects the first image to be most underfocused, and last to be most overfocused.

Prints progress every 5 seconds, though for reasonable sized datasets (7 x 4096 x 4096) it only takes ~ 7 seconds.

Parameters

- **`stack`** (*`3D array`*) – NumPy array (L x M x N). There are L individual (M x N) images.
- **`defvals`** (*`1D array`*) – Array of length L. The defocus values of the images.
- **`v`** (*`int`*) – Verbosity. Set v=0 to suppress all print statements.

Returns Numpy array (M x N) of derivative values.

Return type `ndarray`

1.2.2 SimLTEM

These two modules, along with `microscopes` from *PyTIE*, calculate the electron phase shift through a given magnetization and simulate the LTEM images.

- `comp_phase` contains the linear superposition method and Mansuripur algorithm for calculating the electron phase shift.
- `sim_helper` has loading, display, and setup functions.

comp_phase module

This module consists of functions for simulating the phase shift of a given object.

It contained two functions:

1) linsupPhi - using the linear superposition principle for application in model based iterative reconstruction (MBIR) type 3D reconstruction of magnetization (both magnetic and electrostatic). This also includes a helper function that makes use of numba and just-in-time (jit) compilation.
2) mansPhi - using the Mansuripur Algorithm to compute the phase shift (only magnetic)

Authors: CD Phatak, Arthur McCray June 2020

`comp_phase.`**`exp_sum`**(*mphi_k*, *ephi_k*, *inds*, *KY*, *KX*, *j_n*, *i_n*, *my_n*, *mx_n*, *Sy*, *Sx*)

Called by linsupPhi when running with multiprocessing and numba.

Numba incorporates just-in-time (jit) compiling and multiprocessing to numpy array calculations, greatly speeding up the phase-shift computation beyond that of pure vectorization and without the memory cost. Running this for the first time each session will take an additional 5-10 seconds as it is compiled.

This function could be further improved by sending it to the GPU, or likely by other methods we haven't considered. If you have suggestions (or better yet, written and tested code) please email amccray@anl.gov.

comp_phase.**linsupPhi**(*mx=1.0*, *my=1.0*, *mz=1.0*, *Dshp=None*, *theta_x=0.0*, *theta_y=0.0*, *pre_B=1.0*, *pre_E=1*, *v=1*, *multiproc=True*)

Applies linear superposition principle for 3D reconstruction of magnetic and electrostatic phase shifts.

This function will take 3D arrays with Mx, My and Mz components of the magnetization, the Dshp array consisting of the shape function for the object (1 inside, 0 outside), and the tilt angles about x and y axes to compute the magnetic and the electrostatic phase shift. Initial computation is done in Fourier space and then real space values are returned.

Parameters

- **mx** (*3D array*) – x component of magnetization at each voxel (z,y,x)
- **my** (*3D array*) – y component of magnetization at each voxel (z,y,x)
- **mz** (*3D array*) – z component of magnetization at each voxel (z,y,x)
- **Dshp** (*3D array*) – Binary shape function of the object. Where value is 0, phase is not computed.
- **theta_x** (*float*) – Rotation around x-axis (degrees). Rotates around x axis then y axis if both are nonzero.
- **theta_y** (*float*) – Rotation around y-axis (degrees)
- **pre_B** (*float*) – Numerical prefactor for unit conversion in calculating the magnetic phase shift. Units 1/pixels^2. Generally (2*pi*b0*(nm/pix)^2)/phi0 , where b0 is the Saturation induction and phi0 the magnetic flux quantum.
- **pre_E** (*float*) – Numerical prefactor for unit conversion in calculating the electrostatic phase shift. Equal to sigma*V0, where sigma is the interaction constant of the given TEM accelerating voltage (an attribute of the microscope class), and V0 the mean inner potential.
- **v** (*int*) – Verbosity. v >= 1 will print status and progress when running without numba. v=0 will suppress all prints.
- **mp** (*bool*) – Whether or not to implement multiprocessing.

Returns Tuple of length 2: (ephi, mphi). Where ephi and mphi are 2D numpy arrays of the electrostatic and magnetic phase shifts respectively.

Return type tuple

comp_phase.**mansPhi**(*mx=1.0, my=1.0, mz=None, beam=[0.0, 0.0, 1.0], thick=1.0, embed=0.0*)

Calculate magnetic phase shift using Mansuripur algorithm [1].

Unlike the linear superposition method, this algorithm only accepts 2D samples. The input given is expected to be 2D arrays for mx, my, mz. It can compute beam angles close to (0,0,1), but for tilts greater than a few degrees (depending on sample conditions) it loses accuracy.

The *embed* argument places the magnetization into a larger array to increase Fourier resolution, but this also seems to introduce a background phase shift into some images. Use at your own risk.

Parameters

- **mx** (*2D array*) – x component of magnetization at each pixel.
- **my** (*2D array*) – y component of magnetization at each pixel.
- **mz** (*2D array*) – z component of magnetization at each pixel.
- **beam** (*list*) – Vector direction of beam [x,y,z]. Default [001].
- **thick** (*float*) – Thickness of the sample in pixels. i.e. thickness in nm divided by del_px which is nm/pixel.

- **embed** (*int*) – Whether or not to embed the mx, my, mz into a larger array for Fourier-space computation. In theory this improves edge effects at the cost of reduced speed, however it also seems to add a background phase gradient in some simulations.

embed value	effect
0	Do not embed (default)
1	Embed in (1024, 1024) array
x (int)	Embed in (x, x) array.

Returns 2D array of magnetic phase shift

Return type ndarray

References

1) Mansuripur, M. Computation of electron diffraction patterns in Lorentz electron microscopy of thin magnetic films. J. Appl. Phys. 69, 5890 (1991).

sim_helper module

1.2.3 GUI

The following modules contain the code that makes a graphical user interface (GUI) for the PyLorentz software. Along with the PyTIE functionalities, the GUI also incorporates image registration to align experimental data for TIE reconstruction.

PyLorentz_GUI module

gui_layout module

gui_styling module

Functions for GUI styling.

In addition to the layout file, these functions focus on the style of the elements composing the GUI.

AUTHOR: Timothy Cote, ANL, Fall 2019.

class gui_styling.**WindowStyle**(*theme_background_color: str*)

Bases: object

The WindowStyle class sets the styling of the window and most elements.

DEF_BACKGROUND
The theme background color.

fonts
The dictionary of font styles.

DEF_FONT
The default body text font.

window_height
Window height.

window_width
Window width.

tab_size
The tab size.

small_tab_size
The small tab size.

styles(*key: str*, *default_folder: Optional[str] = None*) → Dict[KT, VT]
The styles within the GUI.

Parameters

- **key** – The key for a PySimpleGUI element.
- **default_folder** – The default folder to browse as defined by defaults.txt.

Returns

A dictionary of the key-value pairs for a specific type of element given by 'key'.

Return type key_style

gui_styling.**get_icon**() → bytes
Returns the icon for display in the dock.

gui_styling.**pad**(*left: int*, *right: int*, *top: int*, *bottom: int*) → Tuple[Tuple[int, int], Tuple[int, int]]
Set padding of element.

Args: left: Left padding. right: Right padding. top: Top padding. bottom: Bottom padding.

Returns padding

Return type The padding format for PySimpleGUI/Tkinter

util module

1.3 Features

PyTIE

- Uses inverse Laplacian method to solve the Transport of Intensity Equation (TIE)
- Can reconstruct the magnetization from samples of variable thickness by using two through focal series (tfs) taken with opposite electron beam directions. *[1]*
- Samples of uniform thickness can be reconstructed from a single tfs.
- Thin samples of uniform thickness, from which the only source of contrast is magnetic Fresnel contrast, can be reconstructed with a single defocused image using Single-Image-TIE (SITIE).
  - This method does not apply to all samples; for more information please refer to Chess et al. *[2]*.
- The TIE and SITIE solvers can implement Tikhonov regularization to remove low-frequency noise *[1]*.
  - Results reconstructed with a Tikhonov filter are no longer quantitative, but a Tikhonov filter can greatly increase the range of experimental data that can be reconstructed.
- Symmetric extensions of the image can be created to reduce Fourier processing edge-effects *[3]*.
- Subregions of the images can be selected interactively to improve processing time or remove unwanted regions of large contrast (such as the edge of a TEM window) that would otherwise interfere with reconstruction.

  - At large aspect ratios, Fourier sampling effects become more pronounced and directionally affect the reconstructed magnetization. Therefore non square images are not quantitative, though symmetrizing the image can greatly reduce this effect.

SimLTEM

- Easily import .omf and .ovf file outputs from OOMMF and Mumax.
- Calculate electron phase shift through samples of a given magnetization with either the Mansuripur algorithm or linear superposition method.
- Simulate LTEM images from these phase shifts and reconstruct the magnetic induction for comparison with experimental results.
- Automate full tilt series for tomographic reconstruction of 3D magnetic samples.
- PyLorentz code is easily integrated into existing python work-flows.

GUI

- TIE reconstruction through a graphical user interface (GUI)
- Additional features include improved region selection and easily images before saving.

1.4 TIE References

(1) Humphrey, E., Phatak, C., Petford-Long, A. K. & De Graef, M. Separation of electrostatic and magnetic phase shifts using a modified transport-of-intensity equation. Ultramicroscopy 139, 5–12 (2014).

(2) Chess, J. J. et al. Streamlined approach to mapping the magnetic induction of skyrmionic materials. Ultramicroscopy 177, 78–83 (2018).

(3) Volkov, V. V, Zhu, Y. & Graef, M. De. A New Symmetrized Solution for Phase Retrieval using the TI. 33, 411–416 (2002).

CHAPTER 2

Getting Started with the GUI

This manual assumes you have already installed the packages and dependencies for PyLorentz. If you have not, follow the instructions *here*. The getting started page helps explain:

Activation of the conda environment for PyLorentz

Layout of the image directory for alignment and reconstruction

Setup of the *.fls file(s)

2.1 Running the GUI

1. Navigate to the `/PyLorentz/GUI` folder.
2. Run `$ conda activate <PyLorentz_Env>`
3. Run `$ python PyLorentz_GUI.py`

The GUI may take a few moments to begin, especially on the first run.

__*Note for Mac Users__

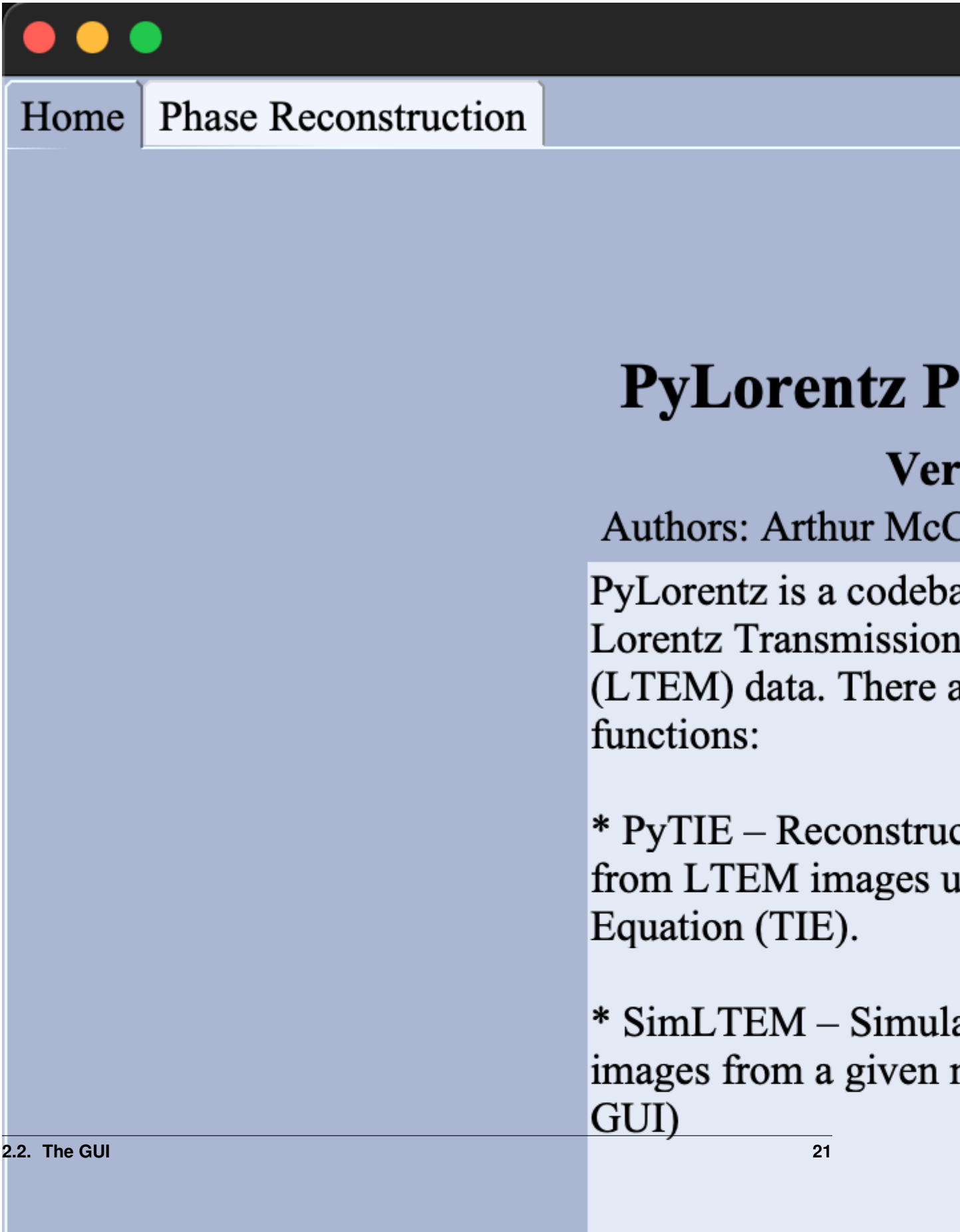
Finder may jump to the front of the screen if it is open. This is ok, it is a process that enables the menu options of PyLorentz specifically for Mac that are inherently capable on Windows.

2.2 The GUI

2.2.1 Menu

The menu has links to opening the main PyLorentz documentation, gives access to opening and closing the log, and has a link to opening this manual.

2.2.2 Home


Home
Phase Reconstruction
PyLorentz P
Ver
Authors: Arthur McC
PyLorentz is a codeba
Lorentz Transmission
(LTEM) data. There a
functions:
* PyTIE – Reconstruc
from LTEM images u
Equation (TIE).
* SimLTEM – Simula
images from a given r
GUI)

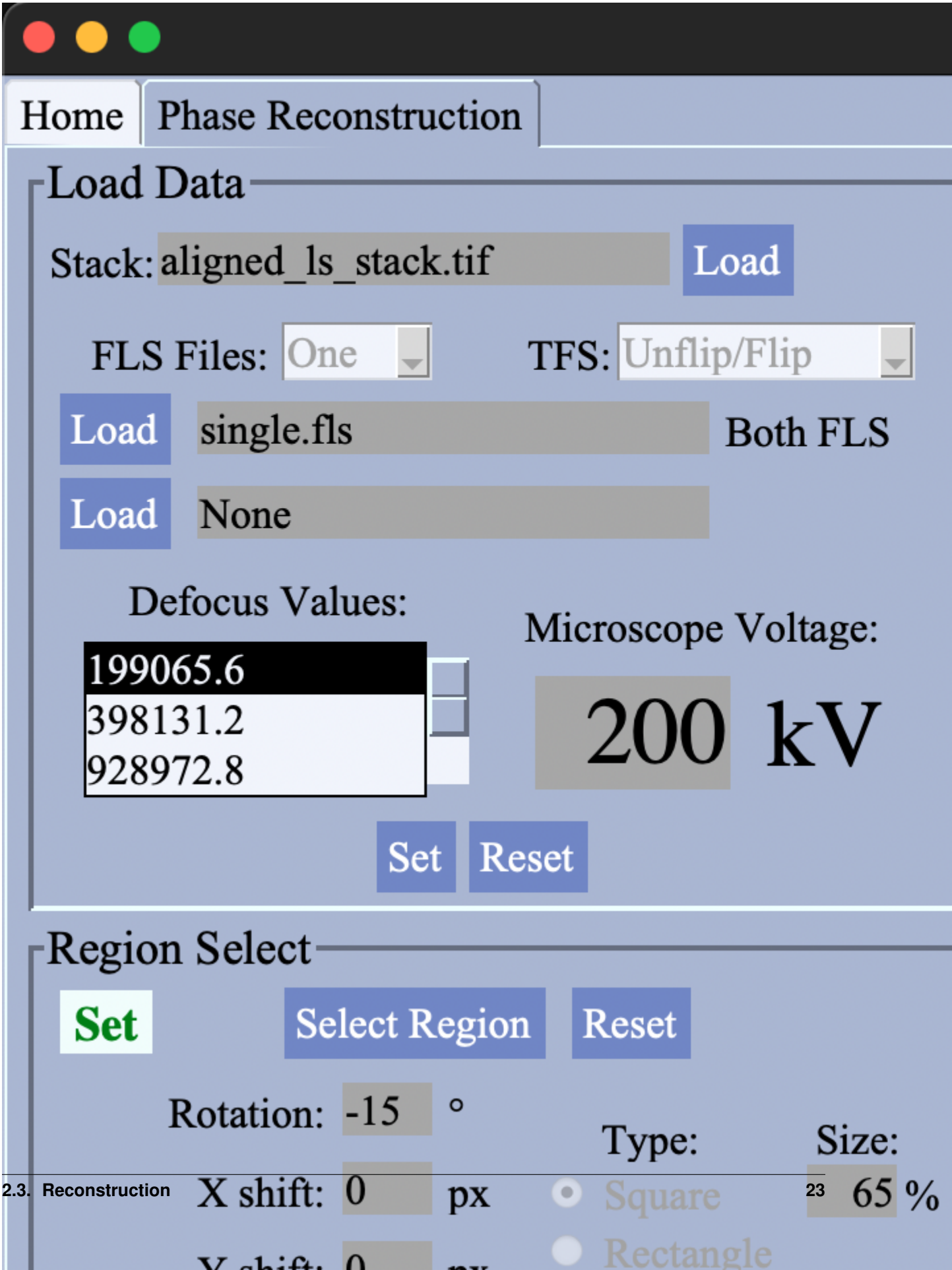
Setting Defaults

- `Browse (2)` Navigate to and select an image working directory.
- `Set as Default (2)` Saves the working directory path in the `defaults.txt` file. On subsequent uses of the GUI, the default working directory input field will be filled as well as the working directory input fields for the `Registration` and `Reconstruction` tabs.
- `Reset Default (2)` Clears the working directory from the `defaults.txt` file.

2.3 Reconstruction

The GUI version of the jupyter notebook meant for solving the transport of intensity equation (TIE) and finding magnetic induction from an experimental through focal series (tfs) of TEM images.

Compared to the notebeook, the GUI has better capabilities of manipulating the ROI for the returned reconstructed images and allows easier viewing of the images as well. The capabilities of the reconstruction itself are the same.


Home
Phase Reconstruction
Load Data
Stack: aligned_ls_stack.tif
Load
FLS Files: One
TFS: Unflip/Flip
Load
single.fls
Both FLS
Load
None
Defocus Values:
199065.6
398131.2
928972.8
Microscope Voltage:
200 kV
Set
Reset
Region Select
Set
Select Region
Reset
Rotation: -15 °
Type:
Size:
X shift: 0 px
Square
65 %
Rectangle

HOW TO USE: 1. Set the image working directory. This will allow access to the FLS Frame and `Set FLS`. 2. Load an aligned image stack. 3. Load *.fls files. Choose whether to perform a double-sided (unflip/flip) tfs alignment or a single-sided tfs alignment (tfs). Then select the number of *.fls files to use. When *.fls file(s) have been chosen, `Set FLS` will become enabled. - If one *.fls file is used for a double sided alignement, the file names must be the same in both the unflip and flip folders. - The *.fls file and the tfs selection should match the stack. Ie. loading a stack aligned with bUnwarpJ won't work if the tfs value is 'Single'. Any errors with selection of stack and *.fls files will appear in the `Log`. - Sometimes the file won't register when loading is selected, try selecting it again if it doesn't appear after selection. 4. Set the accelerating voltage for the microscope, and click `Set FLS`. This will initialize the PTIE parameters. 5. Once initialization is complete, the ROI of the stack will be reduced as only areas common to all images will appear in a reconstructed image. 6. (Optional) Adjust the selected region of interest by clicking `Select Region`. A mask can be applied along with rotations/translations to the images in the stack. The size of the mask can be adjusted and it can be moved around the display by dragging the center of the mouse on the display where the center of the mask should be. Reset any changes using `Reset`. If a mask region is selected, the ROI on display in the graph will be reduced, indicating this will be the region for reconstruction.

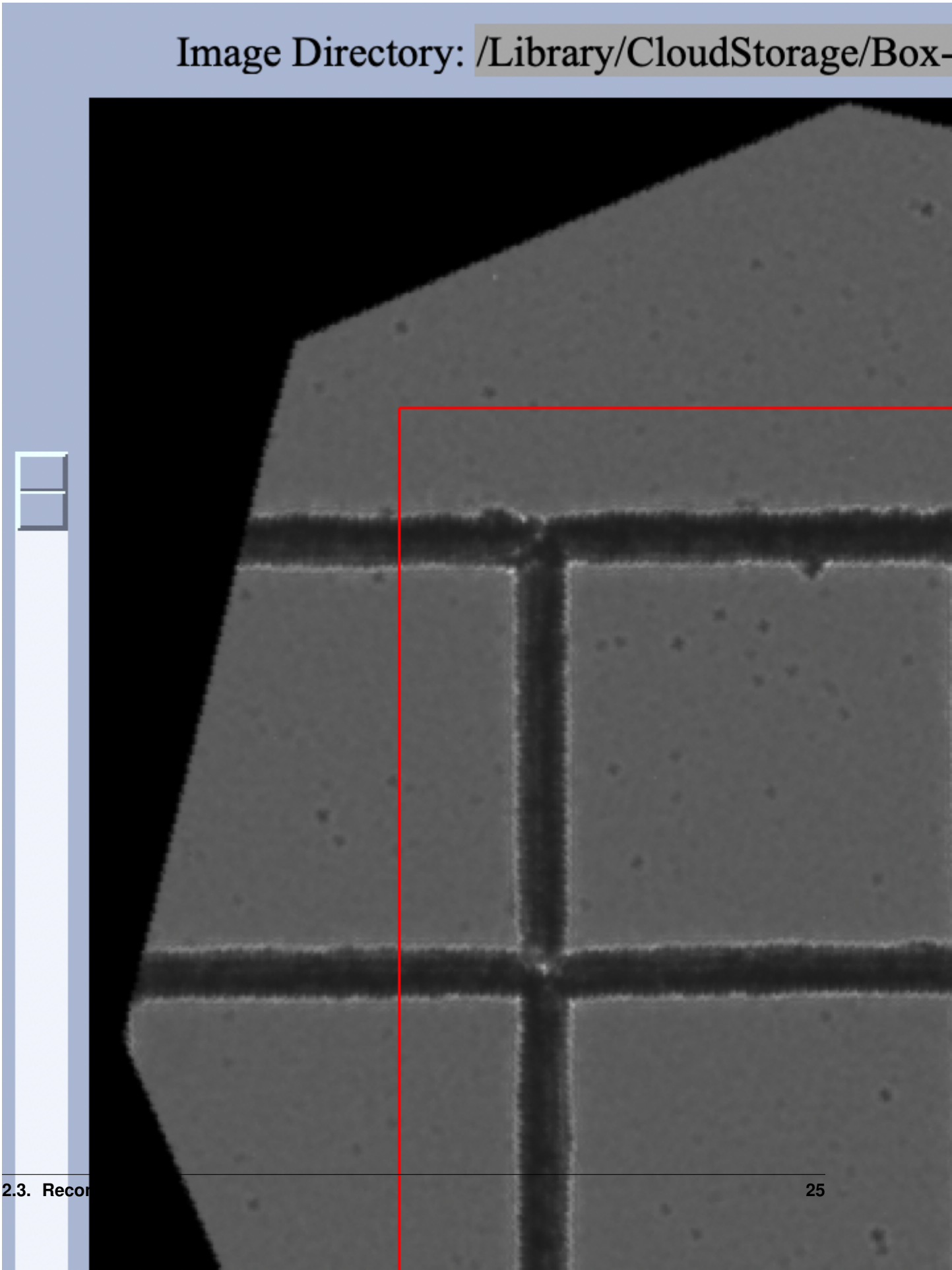


Image Directory: /Library/CloudStorage/Box-

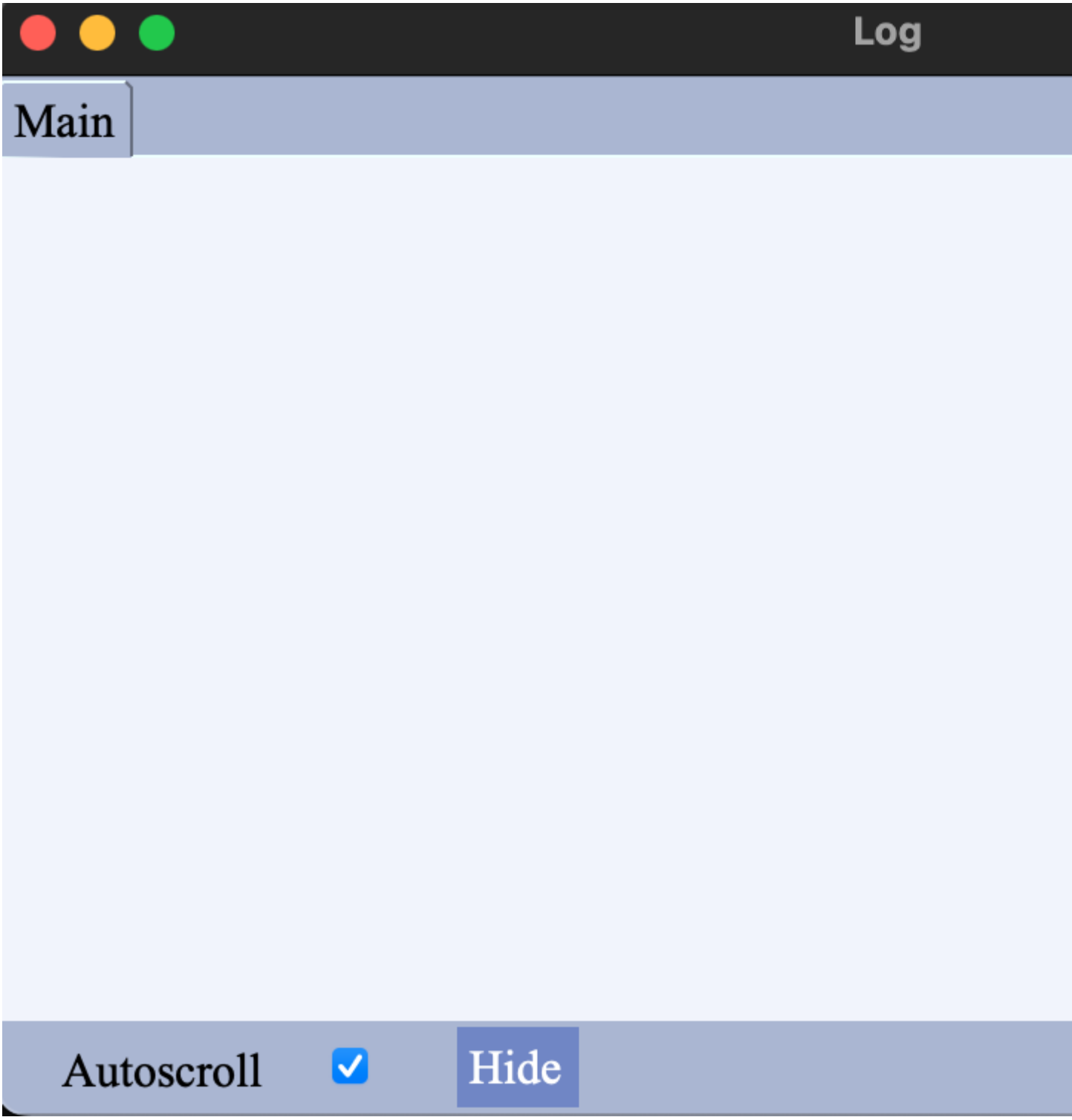
7. Choose the parameters for the reconstruction. Select betweehn the different defocus values, the QC value, symmetrization, the type of the colorwheel, etc. When satisfied click `Run`. After a few moments the images will be loaded and can ve selected in the Image Choices box. View the images by selecting those highlighted in green. - Image not highlighted in green can't be viewed. For a single-sided tfs, neither the electrostic phase or electrostatic derivative can be viewed. - Refer to the PyTIE template notebook for or wiki for more information on the reconstructed images. 8. If satisfied with the images, they can be saved. No images will be saved if `Save` is not clicked and the window is exited. If files will be overwritten, it must be agreed to by the user. A prefix for labeling the images can be set. There are different options for saving different images. - The color image only. - The color image, x-magnetic induction, y-magnetic induction and vectorized magnetic saturation plots. - All images will be saved. - No imags will be saved.

2.4 Log

The log keeps track of any output, misclicks or errors that occur while using the GUI. It is useful for keeping track of which tabs have been entered for any documentation of errors by noticing the different prefixs:

```
- HOM: Home Tab
- REC: Reconstruction Tab
```

The log should be the first place ot check if you expected something to happen and don't know why it happened. The log can be accessed from the Menu or by pressing `Control + l` on your computer. It can be easily hidden with `Control + h`.



2.5 Final Notes

- The intensive processes of the GUI are handled in threads, so it is completely possible to run a reconstruction while running an alignment.

- The Log can serve as your guide for any issues an error. If you can't solve your error from the Log message, take a look at the contents of your file setup and the contents of the *.fls file. If your problem keeps repeating, log an issue to PyLorentz Github.

CHAPTER 3

References

3.1 Linear Superposition Method

For details regarding the linear superposition method of calculating the electron phase shift through a magnetic sample, please see our paper:

> McCray, A. R. C., Cote, T., Li, Y., Petford-Long, A. K. & Phatak, C. Understanding Complex Magnetic Spin Textures with Simulation-Assisted Lorentz Transmission Electron Microscopy. Phys. Rev. Appl. 15, 044025 (2021).

3.2 LTEM References

For a brief introduction to Lorentz Transmission Electron Microscopy, the following papers and textbooks may be of some assistance:

- Phatak, C., Petford-Long, A. K. & De Graef, M. Recent advances in Lorentz microscopy.
- De Graef, M. & Zhu, Y. Quantitative noninterferometric Lorentz microscopy.
- De Graef, M. Introduction to conventional transmission electron microscopy.
- Reimer, L. & Kohl, H. Transmission Electron Microscopy Physics of Image Formation.
- Williams, D. & Carter, C. Transmission Electron Microscopy.

3.3 Micromagnetics References

The simulation side of PyLorentz begins with the output of micromagnetic simulations. PyLorentz has been tested with OOMMF and Mumax; we recommend you refer to their documentation pages for background and information on setting up your own micromagnetic simulations.

There is also Ubermag, a project that interfaces with both OOMMF and Mumax allowing one to run micromagnetic simulations though Python in Jupyter notebooks. It has helpful display functionalities and makes it easier to begin using micromagnetics without learning the scripting languages.

For those wanting to get started with OOMMF or Ubermag, the Online Spintronics Seminar Series presented a series of video tutorials that are very helpful.

CHAPTER 4

Support, License, How to cite, and Acknowledgements

Support

If you have questions or want to report a bug, please create an issue on our GitHub page or email cd@anl.gov.

License

How to cite

If you use PyLorentz in your work, please cite our paper:

> McCray, A. R. C., Cote, T., Li, Y., Petford-Long, A. K. & Phatak, C. Understanding Complex Magnetic Spin Textures with Simulation-Assisted Lorentz Transmission Electron Microscopy. Phys. Rev. Appl. 15, 044025 (2021).

PyLorentz also has a DOI through Zenodo that can be cited in addition to the paper: link

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Materials Sciences and Engineering Division.

genindex

Python Module Index

Index

Q

R

S

T

W

X

Y