TomoPy Documentation

Release 1.2.1

Argonne National Laboratory

Nov 05, 2018

Contents

TomoPy is an open-source Python package for tomographic data processing and image reconstruction.

- Image reconstruction algorithms for tomography.
- Various filters, ring removal algorithms, phase retrieval algorithms.
- Forward projection operator for absorption and wave propagation.

CHAPTER 1

Contribute

- Issue Tracker: https://github.com/tomopy/tomopy/issues
- Documentation: https://github.com/tomopy/tomopy/tree/master/doc
- Source Code: https://github.com/tomopy/tomopy/tree/master/tomopy
- Tests: https://github.com/tomopy/tomopy/tree/master/test

CHAPTER 2

Table of Contents

2.1 About

Tomographic reconstruction creates three-dimensional views of an object by combining two-dimensional images taken from multiple directions, for example, this is how a CAT (computer-aided tomography) scanner generates 3D views of the heart or brain.

Data collection can be rapid, but the required computations are massive and often the beamline staff can be overwhelmed by data that are collected far faster than corrections and reconstruction can be performed *[C15]*. Further, many common experimental perturbations can degrade the quality of tomographs, unless corrections are applied.

To address the needs for image correction and tomographic reconstruction in an instrument independent manner, the TomoPy code was developed *[A1]*, which is a parallelizable high performance reconstruction code.

2.2 Install directions

This section covers the basics of how to download and install TomoPy.

Contents:

2.2.1 Supported Environments

TomoPy is tested, built, and distributed for python 2.7 3.5 3.6 on Linux/macOS and python 3.5 3.6 on Windows 10.

2.2.2 Installing from Conda (Recommended)

If you only want to run TomoPy, not develop it, then you should install through a package manager. Conda, our supported package manager, can install TomoPy and its dependencies for you.

First, you must have Conda installed, then open a terminal or a command prompt window and run:

```
$ conda install -c conda-forge tomopy
```

This will install TomoPy and all the dependencies from the conda-forge channel.

Updating the installation

TomoPy is an active project, so we suggest you update your installation frequently. To update the installation run:

```
$ conda update -c conda-forge tomopy
```

For some more information about using Conda, please refer to the docs.

2.2.3 Installing from source with Conda

Sometimes an adventurous user may want to get the source code, which is always more up-to-date than the one provided by Conda (with more bugs of course!).

For this you need to get the source from the TomoPy repository on GitHub. Download the source to your local computer using git by opening a terminal and running:

```
$ git clone https://github.com/tomopy/tomopy.git
```

in the folder where you want the source code. This will create a folder called *tomopy* which contains a copy of the source code.

Installing dependencies

You will need to install all the dependencies listed in `requirements.txt` or `meta.yaml` files. For example, requirements can be installed using Conda by running:

```
$ conda install --file requirements.txt
```

After navigating to inside the *tomopy* directory, you can install TomoPy by building/compiling the shared libraries and running the install script:

```
$ python build.py
$ pip install .
```

Common issues

No issues with the current build system have been reported.

2.2.4 Importing TomoPy

When importing, it is best to import TomoPy before importing numpy. See this thread for details.

2.3 Tomographic data files

For reading tomography files formatted in different ways, please go check the DXchange package. There are various examples and demonstration scripts about how to use the package for loading your datasets.

The package can be installed by simply running the following in a terminal:

```
conda install -c conda-forge dxchange
```

For a repository of experimental and simulated data sets please check TomoBank *[C6]*.

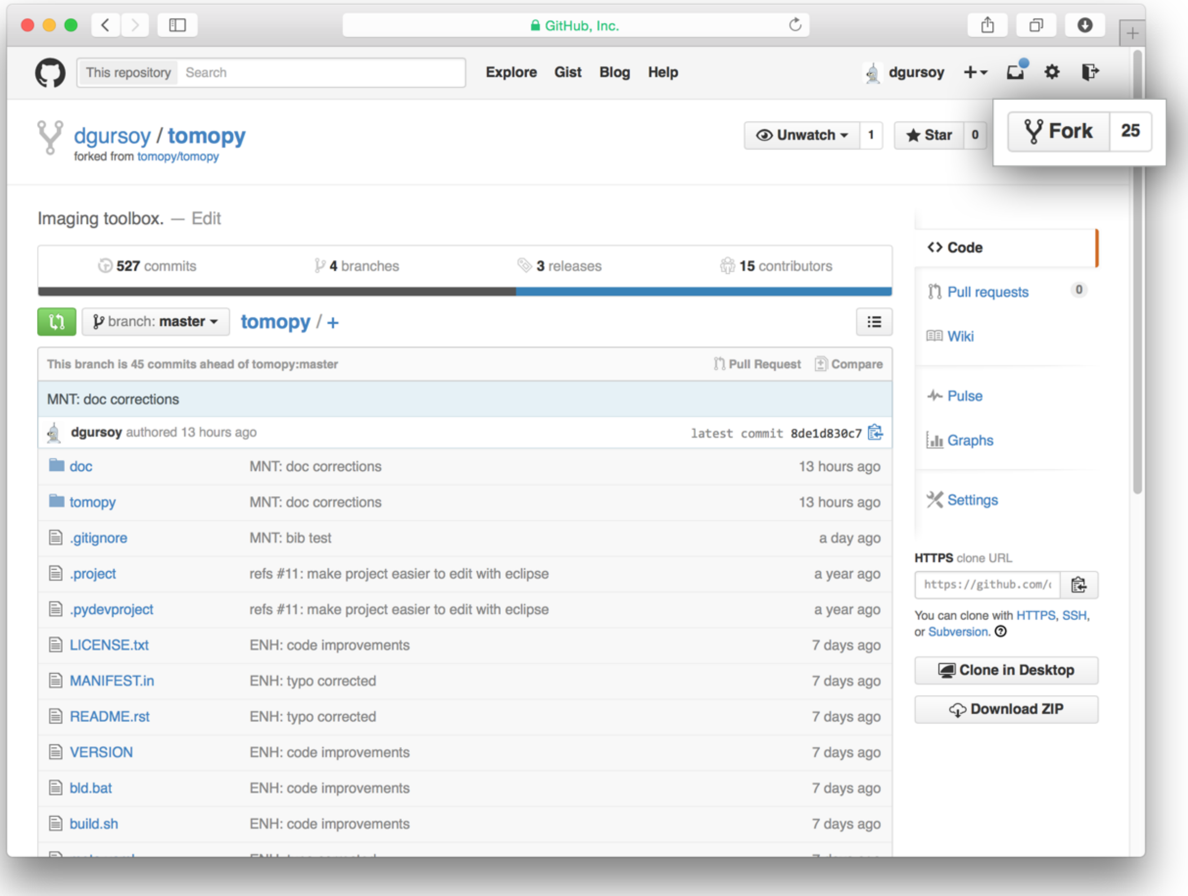
2.4 Development

This section explains the basics for developers who wish to contribute to the TomoPy project.

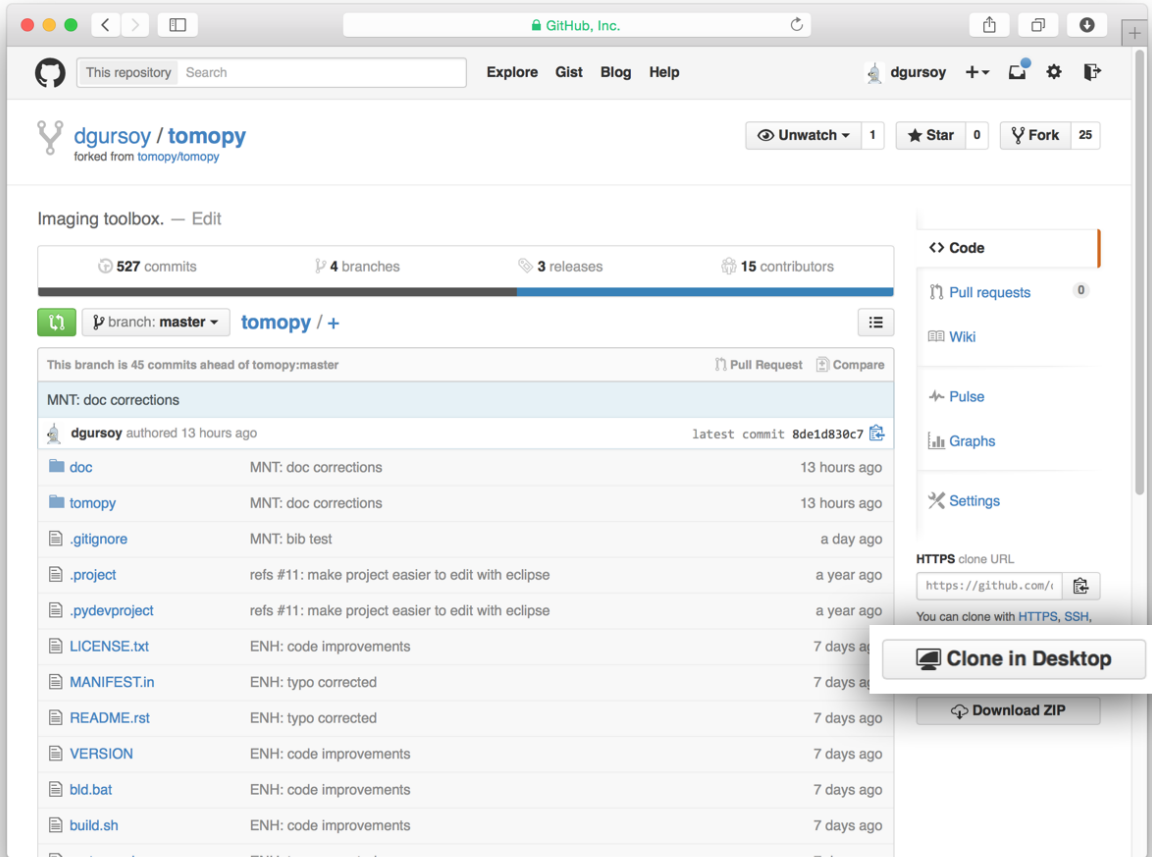
Contents:

2.4.1 Cloning the repository

The project is maintained on GitHub, which is a version control and a collaboration platform for software developers. To start first register on GitHub and fork the TomoPy repository by clicking the **Fork** button in the header of the TomoPy repository:



This creates a copy of the project in your personal GitHub space. The next thing you want to do is to clone it to your local machine. You can do this by clicking the **Clone in Desktop** button in the bottom of the right hand side bar:



This will launch the GitHub desktop application (available for both Mac and Win) and ask you where you want to save it. Select a location in your computer and feel comfortable with making modifications in the code.

2.4.2 Running the Tests

Tomopy has a suite of python unit tests that live in the `/test` directory, where they follow the same tree structure as the packages under `/tomopy`. These are automatically run by TravisCI when you make a pull request (See below for how to do that) and you can run them manually using pytest, or whichever python test runner you prefer. To make it easier to run tests on the changes you make to the code, it is recommended that you install TomoPy in development mode. (*python setup.py develop*)

The pytest test runner, is available through pip or anaconda.

To run the tests open a terminal, navigate to your project folder, then run `py.test`.

To run sections of tests, pass py.test a directory or filepath, as in `py.test test/test_recon` or `py.test test/test_recon/test_rotation.py`.

When writing tests, at minimum we try to check all function returns with synthetic data, together with some dimension, type, etc. Writing tests is highly encouraged!

2.4.3 Coding conventions

We try to keep our code consistent and readable. So, please keep in mind the following style and syntax guidance before you start coding.

First of all the code should be well documented, easy to understand, and integrate well into the rest of the project. For example, when you are writing a new function always describe the purpose and the parameters:

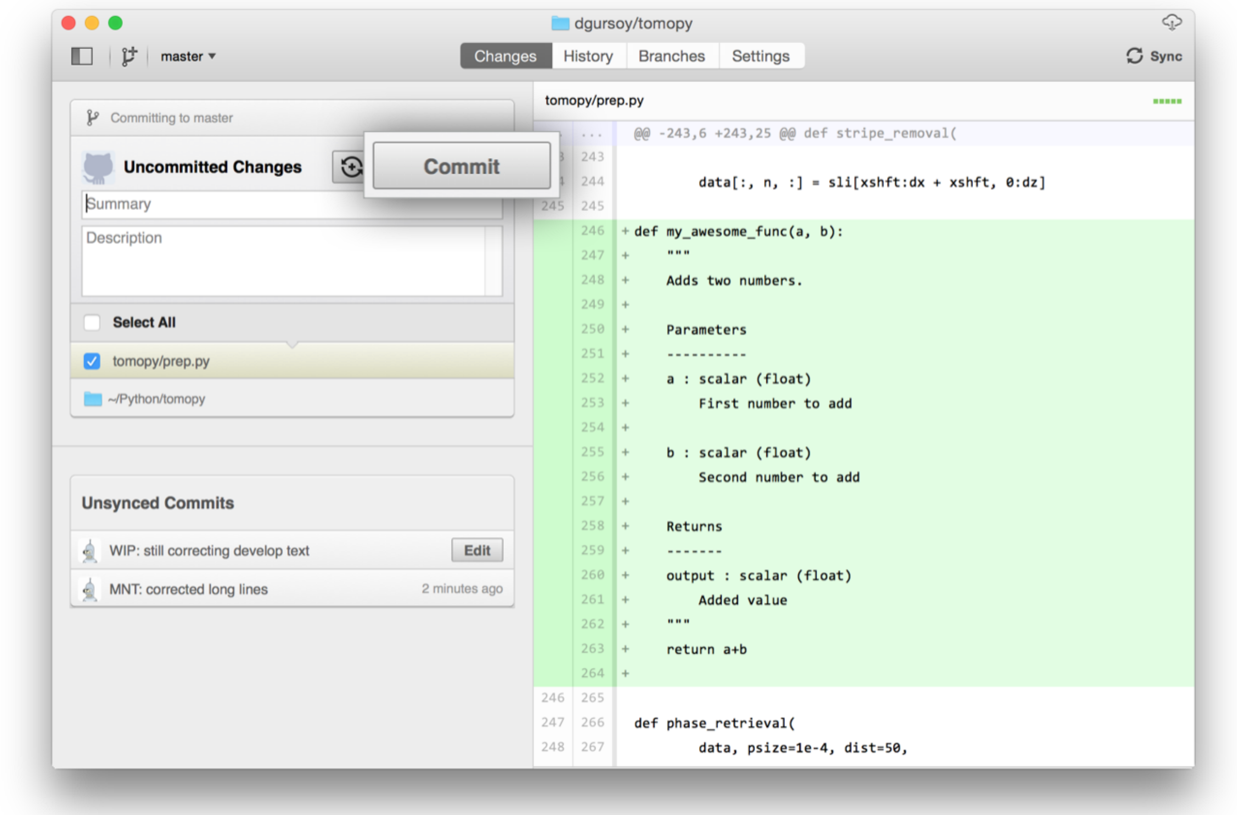
```
def my_awesome_func(a, b):
    """
    Adds two numbers.

    Parameters
    ----------
    a : scalar (float)
        First number to add

    b : scalar (float)
        Second number to add

    Returns
    -------
    output : scalar (float)
        Added value
    """
    return a+b
```

2.4.4 Package versioning

We follow the X.Y.Z (Major.Minor.Patch) semantic for package versioning. The version should be updated before each pull request accordingly. The patch number is incremented for minor changes and bug fixes which do not change the software's API. The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

2.4.5 Committing changes

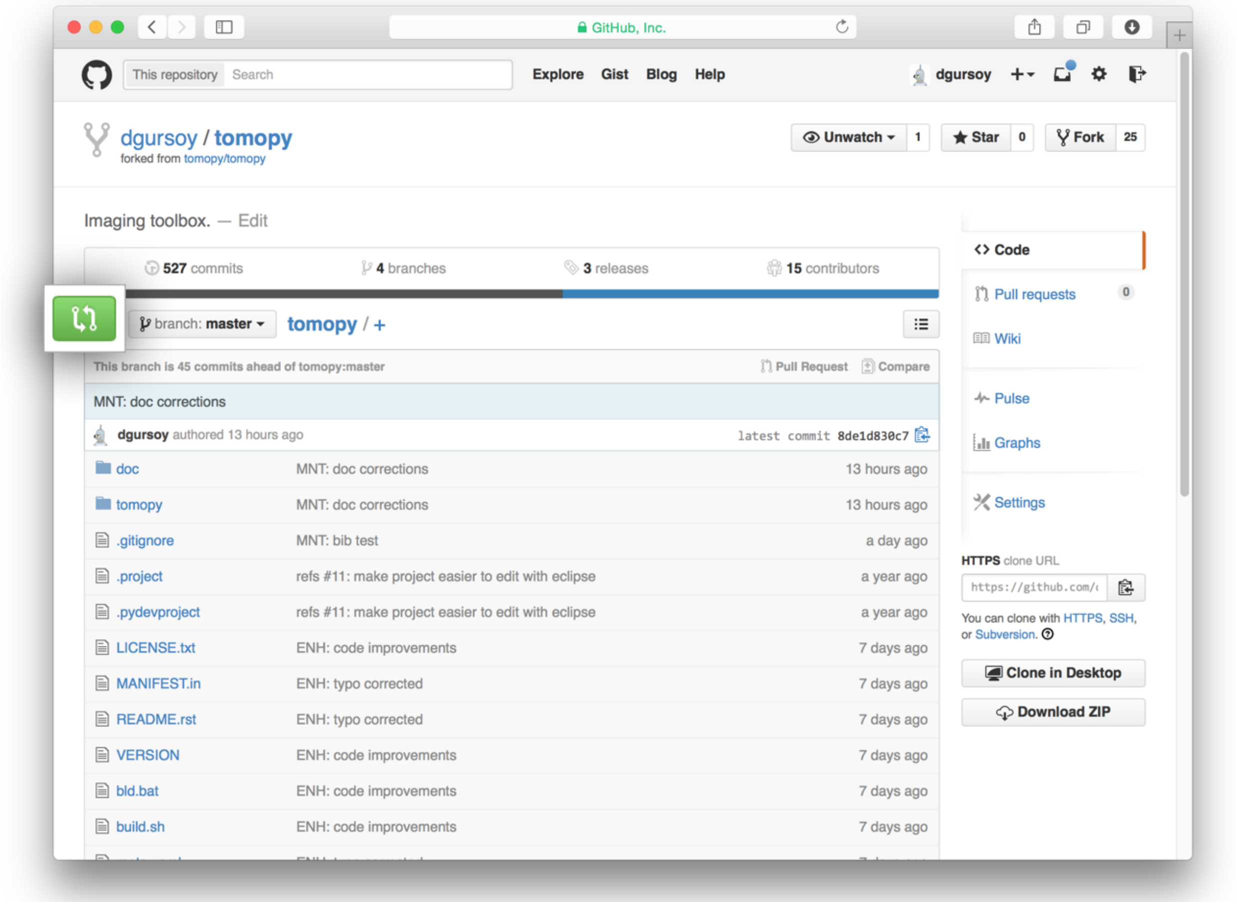
After making some changes in the code, you may want to take a *snapshot* of the edits you made. That's when you make a *commit*. To do this, launch the GitHub desktop application and it should provide you all the changes in your code since your last commit. Write a brief *Summary* and *Description* about the changes you made and click the **Commit** button:



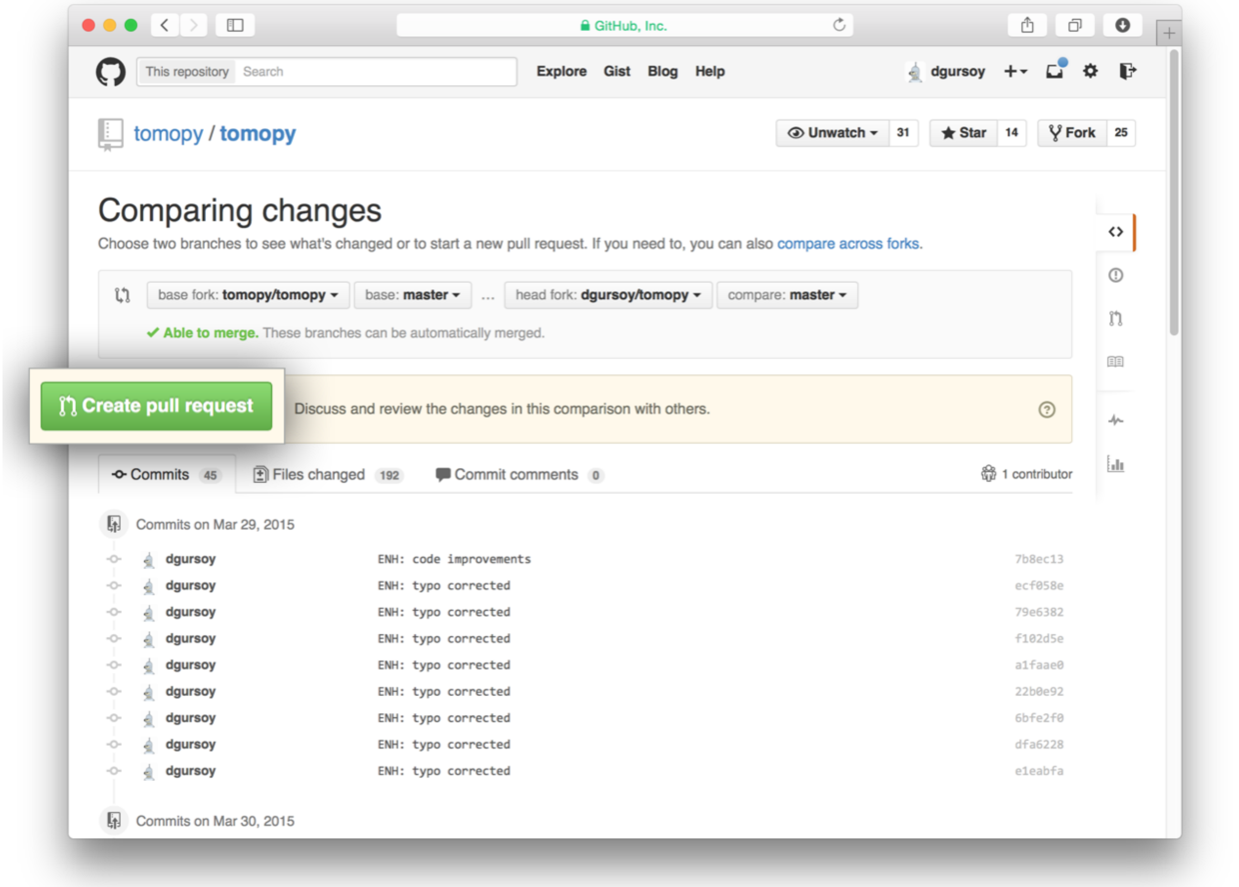
You can continue to make changes, add modules, write your own functions, and take more *Commit snapshots* of your code writing process.

2.4.6 Contributing back

Once you feel that the functionality you added would benefit the community, then you should consider contributing back to the TomoPy project. You will need to push your local commits to GitHub, then go to your online GitHub repository of the project and click on the *green* button to compare, review, and create a pull request.



After clicking on this button, you are presented with a review page where you can get a high-level overview of what exactly has changed between your forked branch and the original TomoPy repository. When you're ready to submit your pull request, click **Create pull request**:



Clicking on **Create pull request** sends you to a discussion page, where you can enter a title and optional description. It's important to provide as much useful information and a rationale for why you're making this Pull Request in the first place.

When you're ready typing out your heartfelt argument, click on **Send pull request**. You're done!

2.5 Release Notes

2.5.1 TomoPy 1.0.0 Release Notes

New features

- FFTW implementation is now adopted. All functions that rely on FFTs such as gridrec, phase retrieval, stripe removal, etc. are now using the FFTW implementation through PyFFTW.
- `sinogram_order` is added to `recon` as an additional argument. It determines whether data is a stack of sinograms (True, y-axis first axis) or a stack of radiographs (False). Default is False, but we plan to make it True in the upcoming release.
- Reconstruction algorithms only copies data if necessary.
- Updated library to support new mproc and recon functions. The data is now passed in sinogram order to recon functions. Also updated tests.
- `ncores` and `nchunks` are now independent.
- Setting `nchunks` to zero removes the dimension. That allows for the functions work on 2D data rather than 3D data.
- Sliced data are used so that each process only receives the data it needs. No more `istart` and `iend` variables for setting up indices in parallel processes.
- Functions will reuse sharedmem arrays if they can.

New functions

- minus_log
- trim_sinogram

New packages in Conda channel

- dxchange 0.1.1
- fftw 3.3.4
- pyfftw 0.9.2
- pywavelets 0.4.0
- xraylib 3.1.0

Deprecated features

- All data I/O related functions are deprecated. They are available through DXchange package.
- Removed fft.h and fft.c, they are now completely replaced with FFTW.

Backward incompatible changes

- `emission` argument is removed from `recon`. After this change the tomographic image reconstruction algorithms always assume data to be normalized.

Contributors

- Arthur Glowacki (@aglowacki)
- Daniel Pelt (@dmpelt)
- Dake Feng (@dakefeng)
- Doga Gursoy (@dgursoy)
- Francesco De Carlo (@decarlof)
- Lin Jiao (@yxqd)
- Luis Barroso-Luque (@lbluque)
- Michael Sutherland (@michael-sutherland)
- Rafael Vescovi (@ravescovi)
- Thomas Caswell (@tacaswell)
- Pete R. Jemian (@prjemian)
- Wei Xu (@celiafish)

2.6 API reference

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

TomoPy Modules:

2.6.1 `tomopy.misc.corr`

2.6.2 `tomopy.misc.morph`

2.6.3 `tomopy.misc.phantom`

2.6.4 `tomopy.prep.alignment`

2.6.5 `tomopy.prep.normalize`

2.6.6 `tomopy.prep.phase`

2.6.7 `tomopy.prep.stripe`

2.6.8 `tomopy.recon.algorithm`

2.6.9 `tomopy.recon.rotation`

2.6.10 `tomopy.sim.project`

2.6.11 `tomopy.sim.propagate`

2.7 Examples

This section contains Jupyter Notebooks and Python scripts examples for various tomoPy functions.

To run these examples in a notebooks install Jupyter or run the python scripts from here

2.7.1 Gridrec

Here is an example on how to use the gridrec *[C5]* reconstruction algorithm with TomoPy *[A1]*. You can download the python scritp `here` or the Jupyter notebook `here`

```
%pylab inline
```

Install TomoPy then:

```
import tomopy
```

Tomographic data input in TomoPy is supported by DXchange.

```
import dxchange
```

matplotlib provide plotting of the result in this notebook. Paraview or other tools are available for more sophisticated 3D rendering.

```
import matplotlib.pyplot as plt
```

Set the path to the micro-CT data to reconstruct.

```
fname = '../../tomopy/data/tooth.h5'
```

Select the sinogram range to reconstruct.

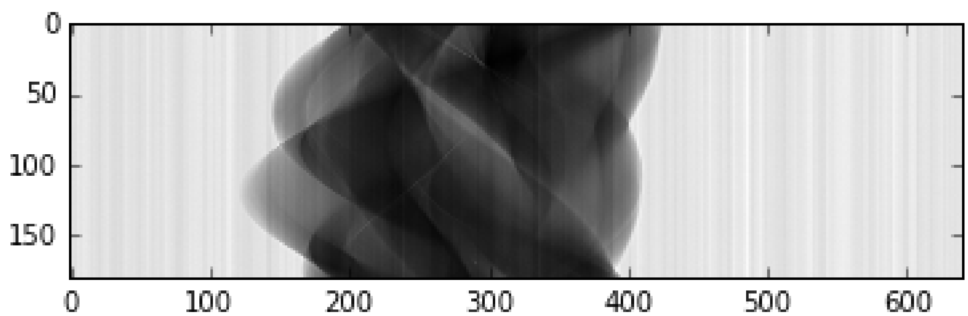
```
start = 0
end = 2
```

`tooth.h5` data set file format follows the APS beamline 2-BM and 32-ID data-exchange file format definition. Major synchrotron file format examples are available at DXchange.

```
proj, flat, dark, theta = dxchange.read_aps_32id(fname, sino=(start, end))
```

Plot the sinogram:

```
plt.imshow(proj[:, 0, :], cmap='Greys_r')
plt.show()
```



If the angular information is not avaialable from the raw data you need to set the data collection angles. In this case theta is set as equally spaced between 0-180 degrees.

```
if (theta is None):
    theta = tomopy.angles(proj.shape[0])
else:
    pass
```

Perform the flat-field correction of raw data:

$$\frac{proj - dark}{flat - dark}$$

```
proj = tomopy.normalize(proj, flat, dark)
```

Tomopy provides various methods (*[C10]*, *[C22]*, *[C14]*) to find the rotation center.

```
rot_center = tomopy.find_center(proj, theta, init=290, ind=0, tol=0.5)
```

```
tomopy.rotation:Trying center: [ 290.]
tomopy.rotation:Trying center: [ 304.5]
tomopy.rotation:Trying center: [ 275.5]
tomopy.rotation:Trying center: [ 282.75]
tomopy.rotation:Trying center: [ 297.25]
tomopy.rotation:Trying center: [ 304.5]
tomopy.rotation:Trying center: [ 304.5]
tomopy.rotation:Trying center: [ 293.625]
```

(continues on next page)

(continued from previous page)

```
tomopy.rotation:Trying center: [ 290.]
tomopy.rotation:Trying center: [ 295.4375]
tomopy.rotation:Trying center: [ 291.8125]
tomopy.rotation:Trying center: [ 294.53125]
tomopy.rotation:Trying center: [ 295.4375]
tomopy.rotation:Trying center: [ 294.078125]
```

Calculate

$$-log(proj)$$

```
proj = tomopy.minus_log(proj)
```

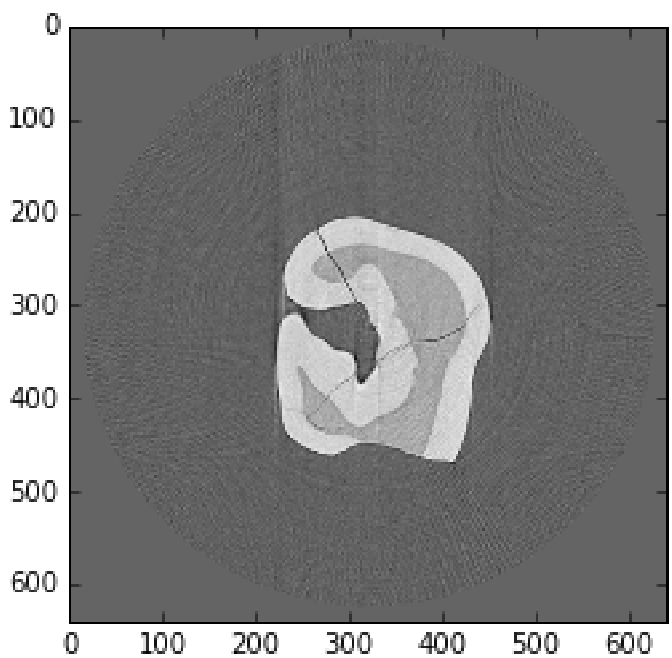
Reconstruction using Gridrec algorithm.Tomopy provides various reconstruction methods including the one part of the ASTRA toolbox.

```
recon = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')
```

Mask each reconstructed slice with a circle.

```
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
```

```
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



2.7.2 Using the ASTRA toolbox through TomoPy

Here is an example on how to use the ASTRA toolbox through its integration with TomoPy, as published in *[A2]*.

```
%pylab inline
```

Install the ASTRA toolbox and TomoPy then:

```
import tomopy
```

DXchange is installed with tomopy to provide support for tomographic data loading. Various data format from all major synchrotron facilities are supported.

```
import dxchange
```

matplotlib provide plotting of the result in this notebook. Paraview or other tools are available for more sophisticated 3D rendering.

```
import matplotlib.pyplot as plt
```

Set the path to the micro-CT data to reconstruct.

```
fname = '../../tomopy/data/tooth.h5'
```

Select the sinogram range to reconstruct.

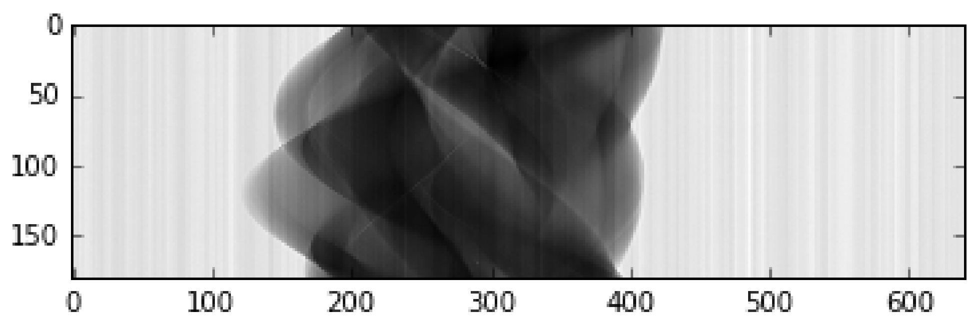
```
start = 0
end = 2
```

This data set file format follows the APS beamline 2-BM and 32-ID definition. Other file format readers are available at DXchange.

```
proj, flat, dark, theta = dxchange.read_aps_32id(fname, sino=(start, end))
```

Plot the sinogram:

```
plt.imshow(proj[:, 0, :], cmap='Greys_r')
plt.show()
```



If the angular information is not avaialable from the raw data you need to set the data collection angles. In this case theta is set as equally spaced between 0-180 degrees.

```
if (theta is None):
    theta = tomopy.angles(proj.shape[0])
else:
    pass
```

Perform the flat-field correction of raw data:

$$\frac{proj - dark}{flat - dark}$$

```
proj = tomopy.normalize(proj, flat, dark)
```

Tomopy provides various methods to find rotation center.

```
rot_center = tomopy.find_center(proj, theta, init=290, ind=0, tol=0.5)
```

Calculate

$$-log(proj)$$

```
proj = tomopy.minus_log(proj)
```

Reconstruction with TomoPy

Reconstruction can be performed using either TomoPy's algorithms, or the algorithms of the ASTRA toolbox.

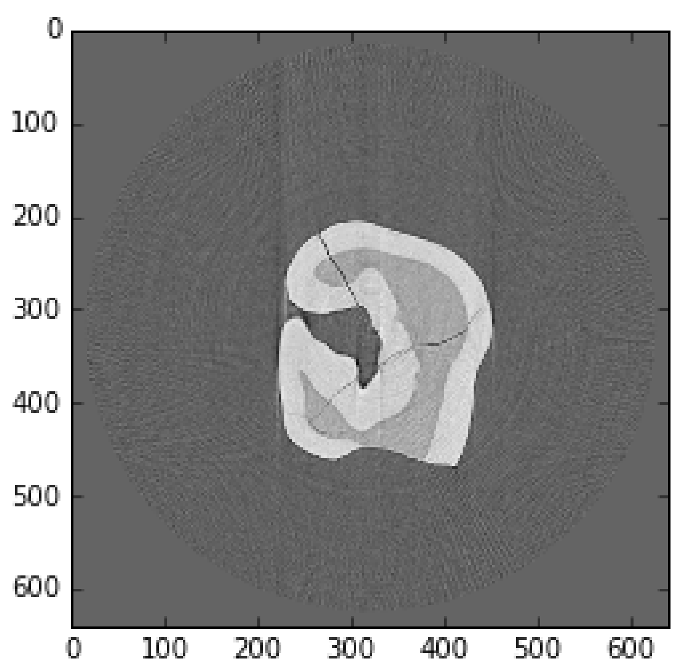
To compare, we first show how to reconstruct an image using TomoPy's Gridrec algorithm:

```
recon = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')
```

Mask each reconstructed slice with a circle.

```
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
```

```
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



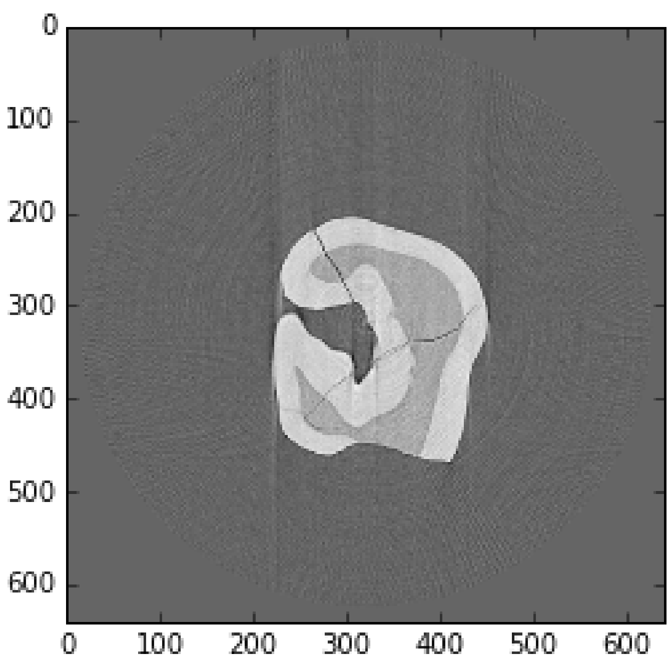
Reconstruction with the ASTRA toolbox

To reconstruct the image with the ASTRA toolbox instead of TomoPy, change the `algorithm` keyword to `tomopy.astra`, and specify the projection kernel to use (`proj_type`) and which ASTRA algorithm to reconstruct with (`method`) in the `options` keyword.

More information about the projection kernels and algorithms that are supported by the ASTRA toolbox can be found in the documentation: projection kernels and algorithms. Note that only the 2D (i.e. slice-based) algorithms are supported when reconstructing through TomoPy.

For example, to use a line-based CPU kernel and the FBP method, use:

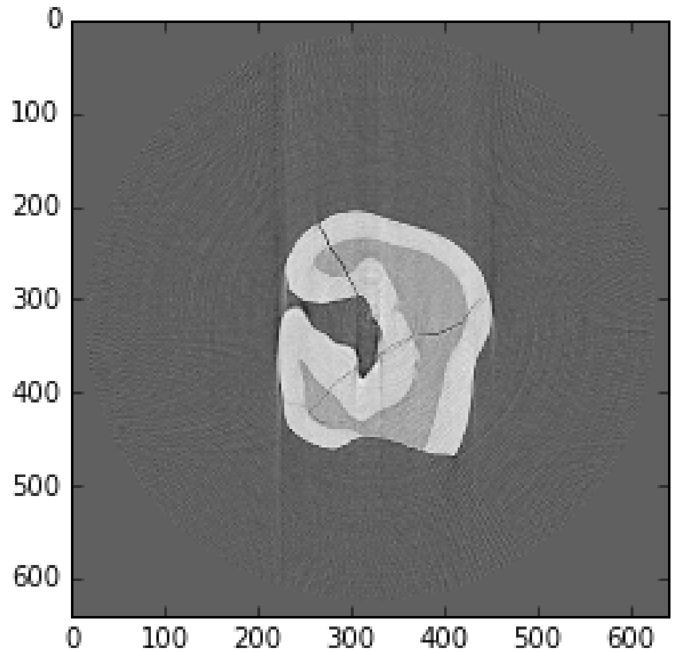
```
options = {'proj_type':'linear', 'method':'FBP'}
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.astra,␣
↪options=options)
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



If you have a CUDA-capable NVIDIA GPU, reconstruction times can be greatly reduced by using GPU-based algorithms of the ASTRA toolbox, especially for iterative reconstruction methods.

To use the GPU, change the `proj_type` option to `'cuda'`, and use CUDA-specific algorithms (e.g. `'FBP_CUDA'` for FBP):

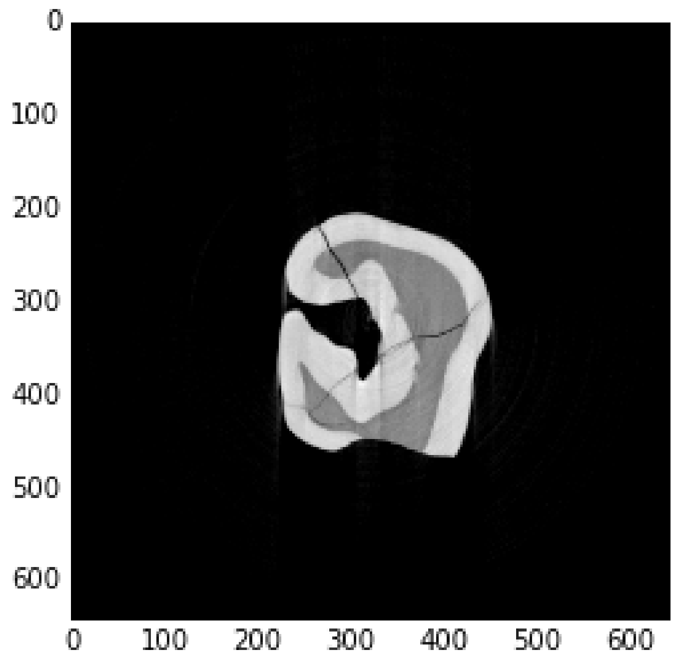
```
options = {'proj_type':'cuda', 'method':'FBP_CUDA'}
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.astra,␣
↪options=options)
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



Many algorithms of the ASTRA toolbox support additional options, which can be found in the documentation. These options can be specified using the `extra_options` keyword.

For example, to use the GPU-based iterative SIRT method with a nonnegativity constraint, use:

```
extra_options ={'MinConstraint':0}
options = {'proj_type':'cuda', 'method':'SIRT_CUDA', 'num_iter':200, 'extra_options
↪':extra_options}
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.astra,␣
↪options=options)
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



2.7.3 Reconstruction with UFO

UFO is a general-purpose image processing framework developed at the Karlsruhe Institute of Technology and uses OpenCL to execute processing tasks on multiple accelerator devices such as NVIDIA and AMD GPUs, AMD and Intel CPUs as well as Intel Xeon Phi cards.

Here is an example on how to use TomoPy with UFO and its accompanying reconstruction algorithms.

Install TomoPy, ufo-core and ufo-filters. Make sure to install the Python Numpy interfaces in the `python` subdirectory of ufo-core.

DXchange is installed with tomopy to provide support for tomographic data loading. Various data format from all major synchrotron facilities are supported.

```
import dxchange
```

matplotlib allows us to plot the result in this notebook.

```
import matplotlib.pyplot as plt
```

Set the path to the micro-CT dataset and the sinogram range to reconstruct.

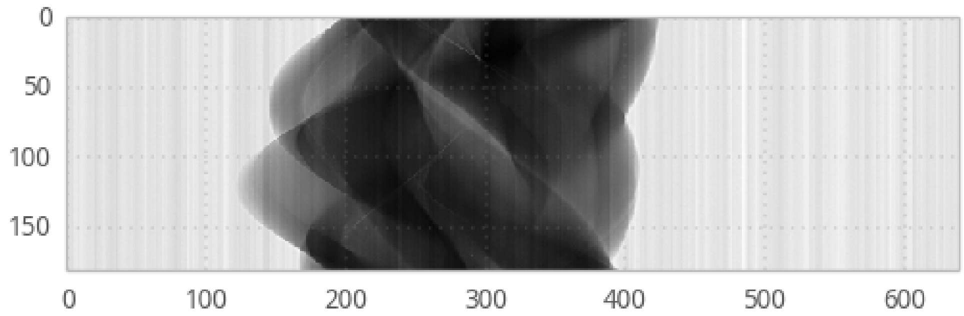
```
fname = 'tooth.h5'
start, end = (0, 2)
```

This dataset file format follows the APS beamline 2-BM and 32-ID definition. Other file format readers are available at DXchange.

```
proj, flat, dark, theta = dxchange.read_aps_32id(fname, sino=(start, end))
```

Plot the sinogram:

```
plt.imshow(proj[:, 0, :], cmap='Greys_r')
plt.show()
```



If the angular information is not available from the raw data you need to set the data collection angles. In this case theta is set as equally spaced between 0-180 degrees.

```
if (theta is None):
    theta = tomopy.angles(proj.shape[0])
else:
    pass
```

Perform the flat-field correction of raw data:

$$\frac{proj - dark}{flat - dark}$$

```
proj = tomopy.normalize(proj, flat, dark)
```

Tomopy provides various methods to find rotation center.

```
center = tomopy.find_center(proj, theta, init=290, ind=0, tol=0.5)
```

Calculate

$$-\log(proj)$$

```
proj = tomopy.minus_log(proj)
```

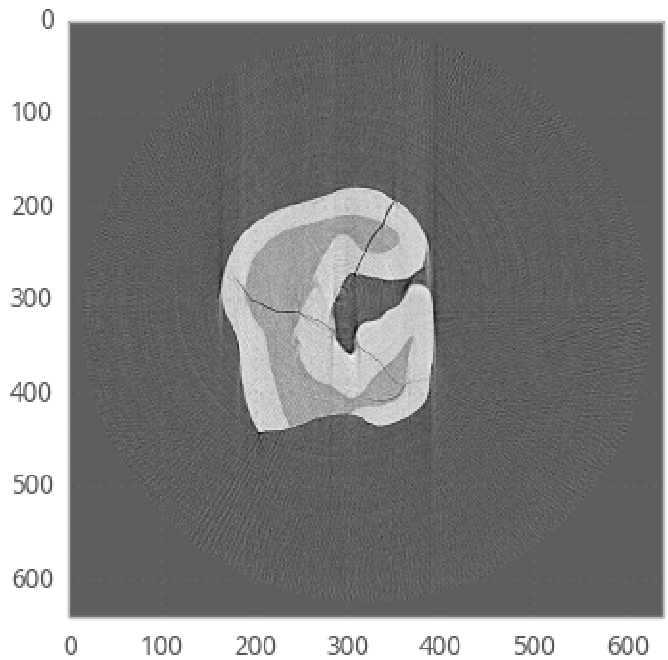
Now, reconstruct using UFO's filtered backprojection algorithm. Note, that we *must* set `ncore` to 1 in order to let UFO do the multi-threading. If left to the default value or set to a value other than 1 will crash the reconstruction.

```
recon = tomopy.recon(proj, theta, center=center, algorithm=ufo_fbp, ncore=1)
```

Mask each reconstructed slice with a circle.

```
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
```

```
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



2.7.4 Vector Reconstruction

The vector reconstruction algorithm can be used for instance, to reconstruct the magnetization vector field inside a magnetic sample.

Here is an example on how to use the vector reconstruction algorithm *[B2]* *[A4]* with TomoPy*[A1]*.

From a reconstructed 3D object to its projections and back

In order to test the algorithm, the projections of a reconstructed oject can be computed, and from these projections we can come back to the reconstructed model object. Finally we will compare the results of the vector field reconstruction against the initial object.

All datasets used in this tutorial are available in tomoBank.

First, let's make the necessary imports

```
import dxchange
import tomopy
import numpy as np
import matplotlib.pyplot as plt
import time
```

Let's load the object: the three components of the magnetization vector all throughout the object. The object will be padded in order to have a cubic object. Afterwards it will be downsampled to make faster computations.

```
obx = dxchange.read_tiff('M4R1_mx.tif').astype('float32')
oby = dxchange.read_tiff('M4R1_my.tif').astype('float32')
obz = dxchange.read_tiff('M4R1_mz.tif').astype('float32')


npad = ((182, 182), (64, 64), (0, 0))
obx = np.pad(obx, npad, mode='constant', constant_values=0)
```

(continues on next page)

(continued from previous page)

```
oby = np.pad(oby, npad, mode='constant', constant_values=0)
obz = np.pad(obz, npad, mode='constant', constant_values=0)

obx = tomopy.downsample(obx, level=2, axis=0)
obx = tomopy.downsample(obx, level=2, axis=1)
obx = tomopy.downsample(obx, level=2, axis=2)

oby = tomopy.downsample(oby, level=2, axis=0)
oby = tomopy.downsample(oby, level=2, axis=1)
oby = tomopy.downsample(oby, level=2, axis=2)

obz = tomopy.downsample(obz, level=2, axis=0)
obz = tomopy.downsample(obz, level=2, axis=1)
obz = tomopy.downsample(obz, level=2, axis=2)
```

Define the projection angles: 31 angles, from 90 to 270 degrees:

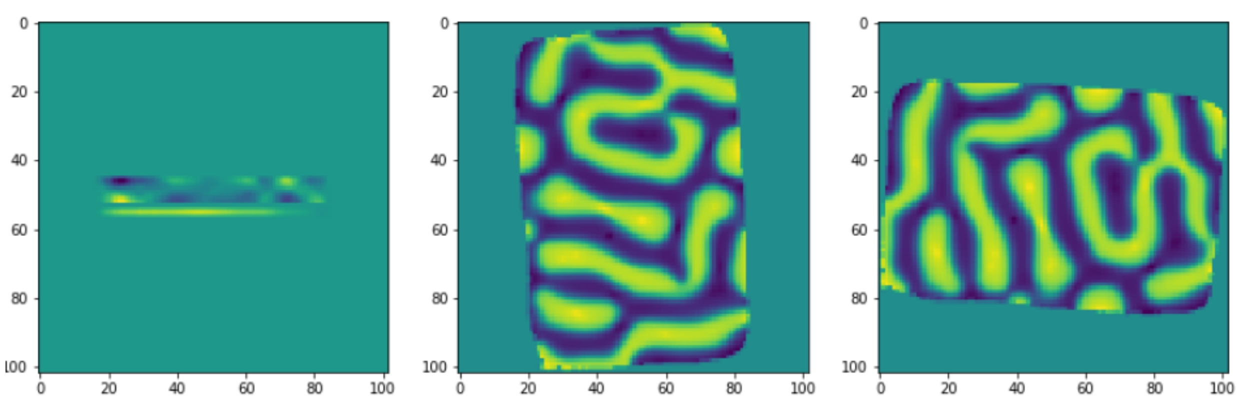
```
ang = tomopy.angles(31, 90, 270)
```

And calculate the projections of the object taking rotation axes around the three perpendicular cartesian axes:

```
prj1 = tomopy.project3(obx, oby, obz, ang, axis=0, pad=False)
prj2 = tomopy.project3(obx, oby, obz, ang, axis=1, pad=False)
prj3 = tomopy.project3(obx, oby, obz, ang, axis=2, pad=False)
```

The three coordinates of a given projection can be visualized as follows:

```
fig = plt.figure(figsize=(15, 8))
ax1 = fig.add_subplot(1, 3, 1)
ax1.imshow(obx[52,:,:])
ax2 = fig.add_subplot(1, 3, 2)
ax2.imshow(oby[52,:,:])
ax3 = fig.add_subplot(1, 3, 3)
ax3.imshow(obz[52,:,:])
```



Finally we will reconstruct the vector field components, taking as input the projections that we have calculated thanks to the first 3D initial object. The number of iterations can be adapted to have a faster but more imprecise reconstruction, or to have a more precise reconstruction.

```
rec1, rec2, rec3 = tomopy.vector3(prj1, prj2, prj3, ang, ang, ang, axis1=0, axis2=1,␣
↪axis3=2, num_iter=100)
```

(continues on next page)

(continued from previous page)

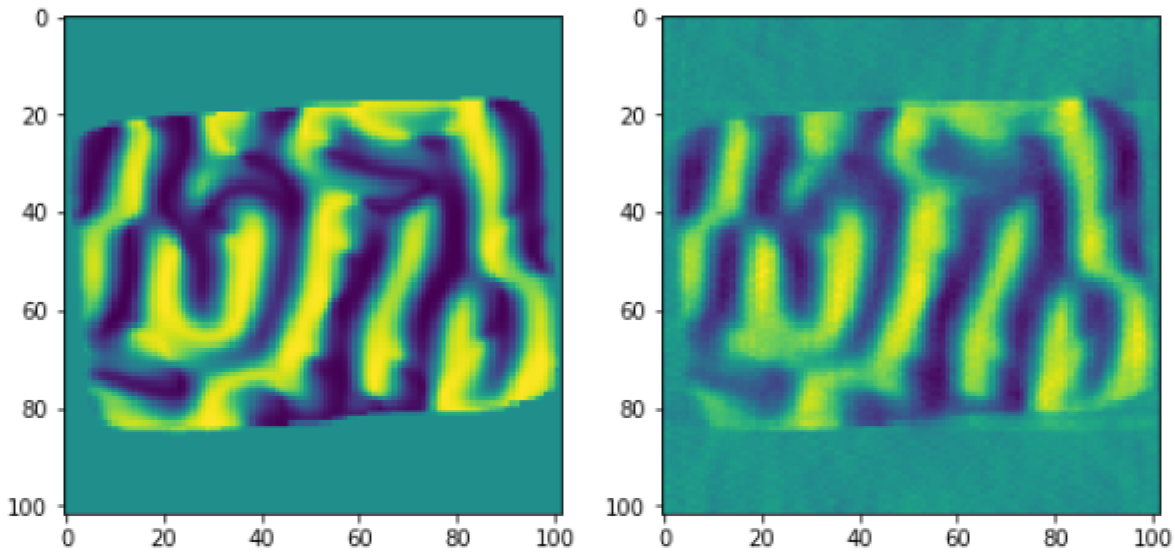
```
dxchange.write_tiff(rec1)
dxchange.write_tiff(rec2)
dxchange.write_tiff(rec3)
```

Comparison of results against input object

In this section, we compare the results of the vector field components obtained thanks to the tomopy reconstruction, against the vector field components of the object given as input:

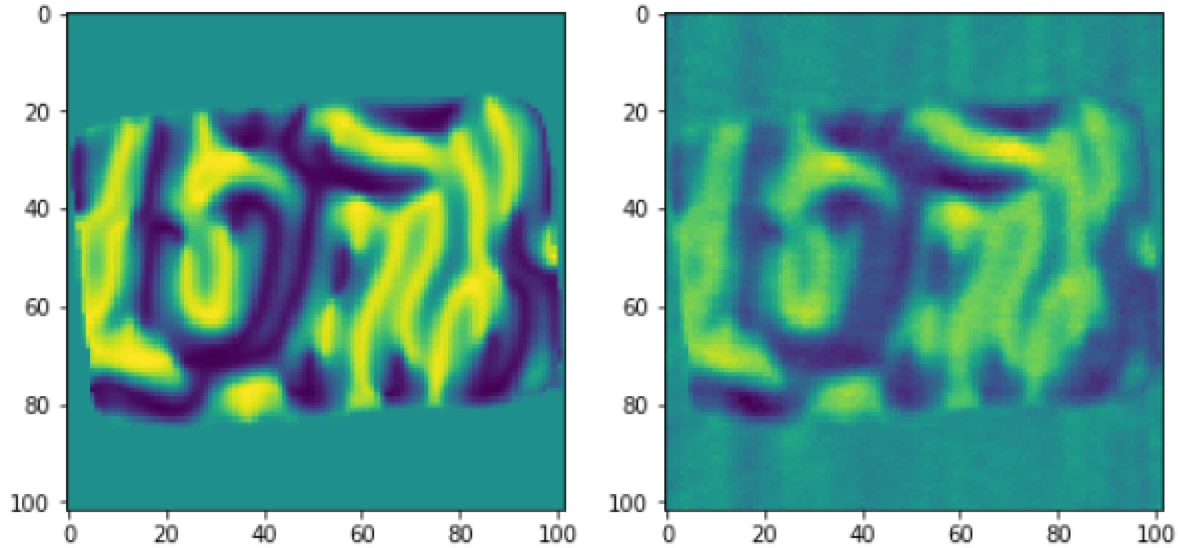
Comparison of the first magnetization vector component against the input data object (for a given slice).

```
fig = plt.figure(figsize=(9, 7))
ax1 = fig.add_subplot(1, 2, 1)
ax1.imshow(obx[52,:,:])
ax2 = fig.add_subplot(1, 2, 2)
ax2.imshow(rec1[52,:,:])
```



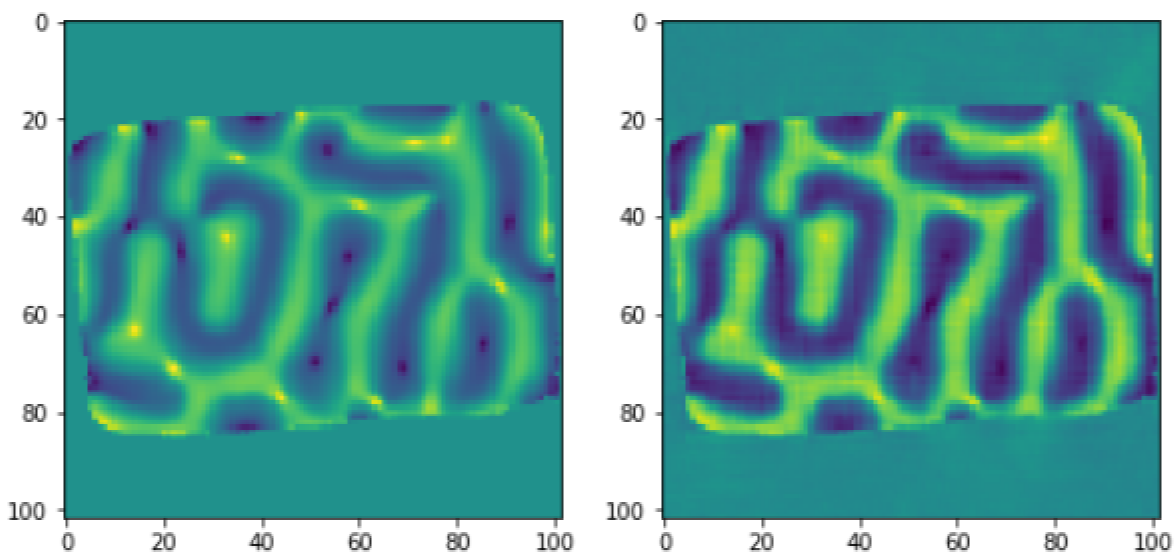
Comparison of the second magnetization vector component against the input data object (for a given slice):

```
fig = plt.figure(figsize=(9, 7))
ax1 = fig.add_subplot(1, 2, 1)
ax1.imshow(oby[52,:,:])
ax2 = fig.add_subplot(1, 2, 2)
ax2.imshow(rec2[52,:,:])
```



Comparison of the third magnetization vector component against the input data object (for a given slice):

```
fig = plt.figure(figsize=(9, 7))
ax1 = fig.add_subplot(1, 2, 1)
ax1.imshow(obz[52,:,:])
ax2 = fig.add_subplot(1, 2, 2)
ax2.imshow(rec3[52,:,:])
```



Other examples

Three jupyter notebooks with examples as well as with some mathematical concepts related to the vector reconstruction, can be found in the tomopy/doc/demo folder:

Examples using vector3: input data projections from 3 orthogonal tilt angles:

- vectorrec_1.ipynb
- vectorrec_disk.ipynb

Example using vector2: input data projections from 2 orthogonal tilt angles:

- vector_heterostructure.ipynb

The Vector Reconstruction examples html slides can be build by applying (from the doc/demo folder) the following commands:

```
jupyter-nbconvert --to slides --post serve vectorrec_1.ipynb
```

```
jupyter-nbconvert --to slides --post serve vectorrec_disk.ipynb
```

```
jupyter-nbconvert --to slides --post serve vector_heterostructure.ipynb
```

2.7.5 LPrec

Here is an example on how to use the log-polar based method (https://github.com/math-vrn/lprec) for reconstruction in Tomopy

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Install lprec from github, then

```
import tomopy
```

DXchange is installed with tomopy to provide support for tomographic data loading. Various data format from all major synchrotron facilities are supported.

```
import dxchange
```

matplotlib provide plotting of the result in this notebook. Paraview or other tools are available for more sophisticated 3D rendering.

```
import matplotlib.pyplot as plt
```

Set the path to the micro-CT data to reconstruct.

```
fname = '../../tomopy/data/tooth.h5'
```

Select the sinogram range to reconstruct.

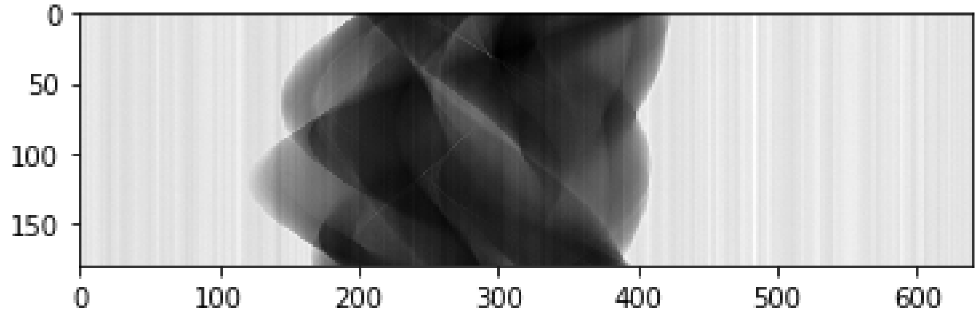
```
start = 0
end = 2
```

This data set file format follows the APS beamline 2-BM and 32-ID definition. Other file format readers are available at DXchange.

```
proj, flat, dark, theta = dxchange.read_aps_32id(fname, sino=(start, end))
```

Plot the sinogram:

```
plt.imshow(proj[:, 0, :], cmap='Greys_r')
plt.show()
```



If the angular information is not avaialable from the raw data you need to set the data collection angles. In this case theta is set as equally spaced between 0-180 degrees.

```
theta = tomopy.angles(proj.shape[0])
```

Perform the flat-field correction of raw data:

$$\frac{proj - dark}{flat - dark}$$

```
proj = tomopy.normalize(proj, flat, dark)
```

Select the rotation center manually

```
rot_center = 296
```

Calculate

$$-log(proj)$$

```
proj = tomopy.minus_log(proj)
```

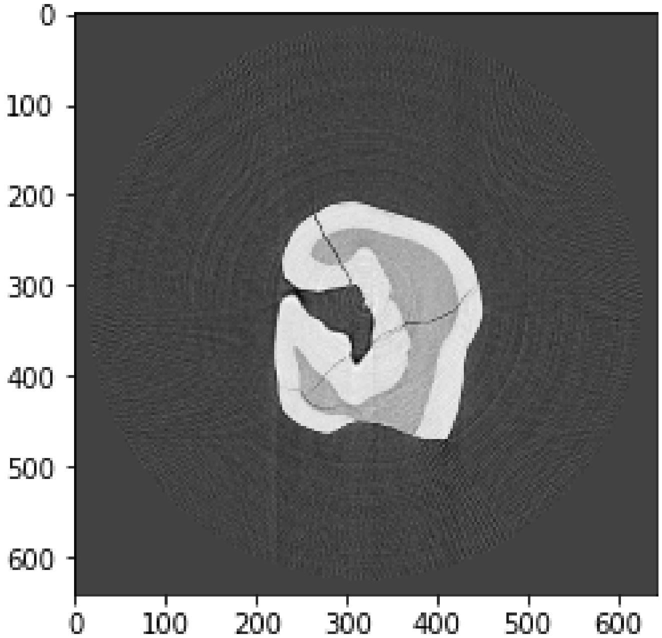
Reconstruction using FBP method with the log-polar coordinates

```
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.lprec, lpmethod=
↪'lpfbp', filter_name='parzen')
```

Mask each reconstructed slice with a circle.

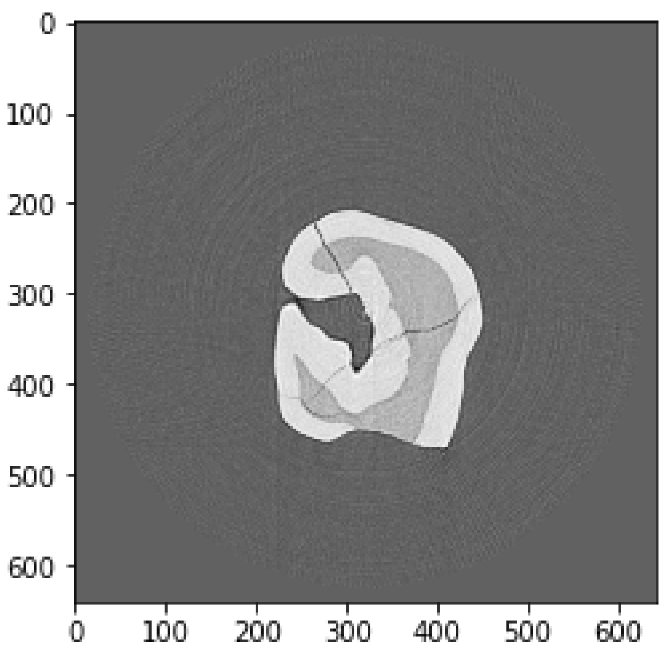
```
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
```

```
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



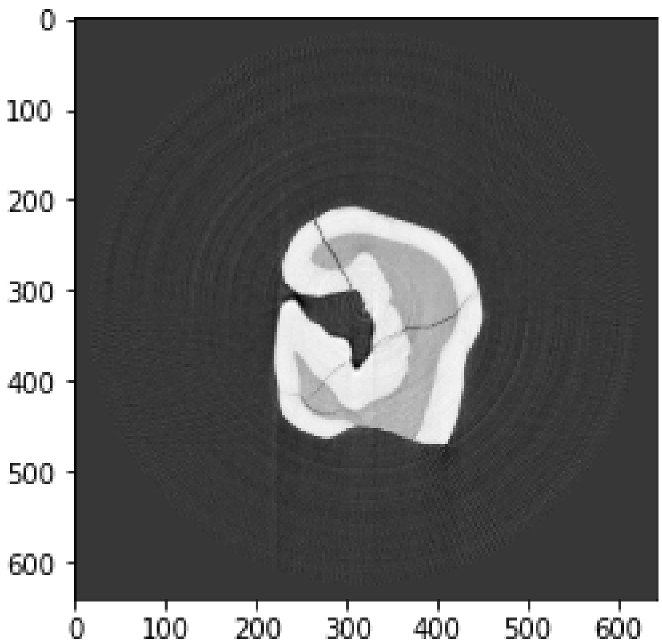
Reconstruction using the gradient descent method with the log-polar coordinates

```
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.lprec, lpmethod=
↪'lpgrad', ncore=1, num_iter=64, reg_par=-1)
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



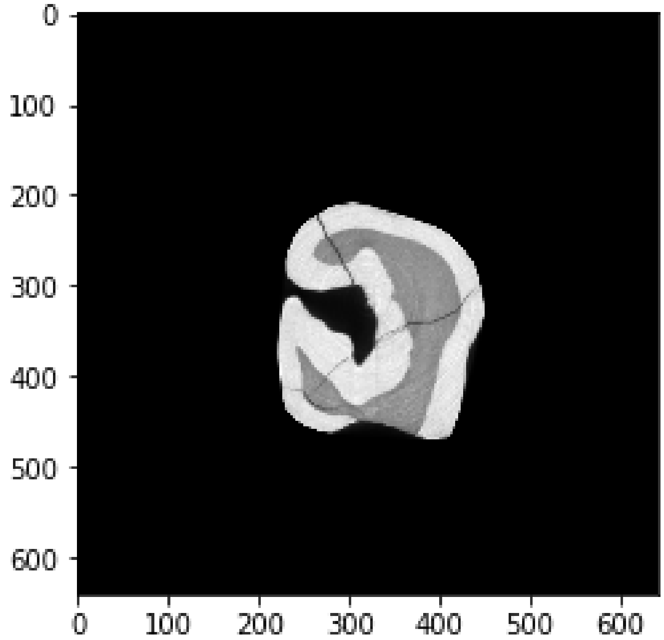
Reconstruction using the TV method with the log-polar coordinates

```
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.lprec, lpmethod=
↪'lptv', ncore=1, num_iter=256, reg_par=2e-4)
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



Reconstruction using the MLEM method with the log-polar coordinates

```
recon = tomopy.recon(proj, theta, center=rot_center, algorithm=tomopy.lprec, lpmethod=
↪'lpem', ncore=1, num_iter=64, reg_par=0.05)
recon = tomopy.circ_mask(recon, axis=0, ratio=0.95)
plt.imshow(recon[0, :,:], cmap='Greys_r')
plt.show()
```



2.8 Frequently asked questions

Here's a list of questions.

Questions

2.8.1 How can I report bugs?

The easiest way to report bugs or get help is to open an issue on GitHub. Simply go to the project GitHub page, click on Issues in the right menu tab and submit your report or question.

2.8.2 Are there any video tutorials?

We currently do not have specific plans in this direction, but we agree that it would be very helpful.

2.8.3 Can I run this on a HPC cluster?

In their default installation packages, TomoPy and the ASTRA toolbox are limited to running on a multicore single machine. The ASTRA toolbox, and TomoPy through the presented ASTRA integration, are able to use multiple GPUs that are installed in a single machine. Both toolboxes can be run on a HPC cluster through parallelization using MPI, but since installation and running on a HPC cluster is often cluster specific, the default installation packages do not include these capabilities.

As such, the integrated packages that is presented in the manuscript currently does not support running on a HPC cluster. Note that the ASTRA toolbox provides a separate MPI enabled package for use on a HPC cluster. We refer to *[C23]* for more details about TomoPy's planned HPC implementation. It is a MapReduce type MPI implementation layer, which was succesfully used on many clusters, i.e. Stampede, Cori, Mira. There are plans to allow user access to TomoPy on a HPC cluster (e.g. through a client or webportal), but these projects will take some time before they are being matured for user's use.

2.8.4 Are there any segmentation routines?

Some data processing operations can be applied after reconstruction. Examples of these type of operations are image based ring removal methods, and gaussian filtering or median filtering the reconstructed image. Typicaly, these methods are called "postprocessing algorithms, since they occur after the reconstruction.

The package does not include segmentation algorithms, since we are currently focused on tomography, while we feel that segmentation are more part of the application specific data analysis that occurs after tomographic processing. An important exception is when segmentation steps are used as part of the tomographic reconstruction algorithm, such as in the DART algorithm.

2.8.5 Are there any tools for aligning projections?

Yes we have. Please check the Examples section for details.

2.8.6 What is ASTRA toolbox?

The ASTRA toolbox provides highly efficient tomographic reconstruction methods by implementing them on graphic processing units (GPUs). It includes advanced iterative methods and allows for very flexible scanning geometries. The ASTRA toolbox also includes building blocks which can be used to develop new reconstruction methods, allowing for easy and efficient implementation and modification of advanced reconstruction methods. However, the toolbox is only focused on reconstruction, and does not include pre-processing or post-processing methods that are typically required for correctly processing synchrotron data. Furthermore, no routines to read data from disk are provided by the toolbox.

2.8.7 Why TomoPy and ASTRA were integrated?

The TomoPy toolbox is specifically designed to be easy to use and deploy at a synchrotron facility beamline. It supports reading many common synchrotron data formats from disk *[C13]*, and includes several other processing algorithms commonly used for synchrotron data. TomoPy also includes several reconstruction algorithms, which can be run on multi-core workstations and large-scale computing facilities. The algorithms in TomoPy are all CPU-based, however, which can make them prohibitively slow in the case of iterative methods, which are often required for advanced tomographic experiments.

By integrating the ASTRA toolbox in the TomoPy framework, the optimized GPU-based reconstruction methods become easily available for synchrotron beamline users, and users of the ASTRA toolbox can more easily read data and use TomoPy's other functionality for data filtering and cleaning.

2.8.8 Which platforms are supported?

TomoPy supports Linux and Mac OS X, and the ASTRA toolbox supports Linux and Windows. As such, the combined package currently supports only Linux, but we are working on supporting more operating systems.

2.8.9 Does ASTRA support all GPUs?

The GPU algorithms are all implemented used nVidia CUDA. As a result, only nVidia CUDA enabled video cards can be used to run them.

2.8.10 What is UFO?

UFO is a general purpose image processing framework, optimized for heterogeneous compute systems and streams of data. Arbitrary data processing tasks are plugged together to form larger processing pipelines. These pipelines are then mapped to the hardware resources available at run-time, i.e. both multiple GPUs and CPUs.

One specific use case that has been integrated into the TomoPy is fast reconstruction using the filtered backprojection and direct Fourier inversion methods although others for pre- and post-processing might be added in the future.

2.9 Credits

We kindly request that you cite the following article(s) *[A1]* if you use TomoPy (and also cite *[A2]* if you use ASTRA or *[A3]* if you use UFO). For vector reconstructions please additionally cite *[A4]*.

2.9.1 Applications

2.9.2 References

CHAPTER 3

License

The project is licensed under the BSD-3 license.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Bibliography

[A1] Gürsoy D, De Carlo F, Xiao X, and Jacobsen C. Tomopy: a framework for the analysis of synchrotron tomographic data. *Journal of Synchrotron Radiation*, 21(5):1188–1193, 2014.

[A2] Pelt D, Gürsoy D, Palenstijn WJ, Sijbers J, De Carlo F, and Batenburg KJ. Integration of tomopy and the astra toolbox for advanced processing and reconstruction of tomographic synchrotron data. *Journal of Synchrotron Radiation*, 23(3):842–849, 2016.

[A3] Vogelgesang M, Chilingaryan S, Rolo T dos Santos, and Kopmann A. Ufo: a scalable gpu-based image processing framework for on-line monitoring. In *Proceedings of The 14th IEEE Conference on High Performance Computing and Communication & The 9th IEEE International Conference on Embedded Software and Systems (HPCC-ICESS)*, 824–829. 6 2012.

[A4] Hierro-Rodriguez A, Gürsoy D, Phatak C, Quiros C, Sorrentino A, Alvarez-Prado LM, Velez M, Martin JI, Alameda JM, Pereiro E, and Ferrer S. 3d reconstruction of magnetization from dichroic soft x-ray transmission tomography. *Journal of Synchrotron Radiation*, 2018.

[B1] Patterson BM, Cordes NL, Henderson K, Williams JJ, Stannard T, Singh SS, Ovejero AR, Xiao X, Robinson M, and Chawla N. In situ x-ray synchrotron tomographic imaging during the compression of hyper-elastic polymeric materials. *Journal of Materials Science*, 51(1):171–187, 2016.

[B2] Phatak C and Gürsoy D. Iterative reconstruction of magnetic induction using lorentz transmission electron tomography. *Ultramicroscopy*, 150:54–64, 2015.

[B3] Gürsoy D, Biçer T, Lanzirotti A, Newville MG, and De Carlo F. Hyperspectral image reconstruction for x-ray fluorescence tomography. *Optics Express*, 23(7):9014–9023, 2015.

[B4] Gürsoy D, Biçer T, Almer JD, Kettimuthu R, Stock SR, and De Carlo F. Maximum a posteriori estimation of crystallographic phases in x-ray diffraction tomography. *Philosophical Transactions A*, 2015.

[B5] Duke DJ, Swantek AB, Sovis N, Tilocco FZ, Powell CF, AL Kastengren, Gürsoy D, and Biçer T. Time-resolved x-ray tomography of gasoline direct injection sprays. *SAE International Journal of Engines*, 2015.

[B6] Kamke FA, McKinley PE, Ching DJ, Zauner M, and Xiao X. Micro x-ray computed tomography of adhesive bonds in wood. *Wood and Fiber Science*, 2016.

[B7] Birkbak ME, Leemreize H, Frohlich S, Stock SR, and Birkedal H. Diffraction scattering computed tomography: a window into the structures of complex nanomaterials. *Nanoscale*, 2015.

[B8] Miller SM, Xiao X, and Faber KT. Freeze-cast alumina pore networks: effects of freezing conditions and dispersion medium. *Journal of the European Ceramic Society*, 35(13):3595—3605, 2015.

[B9] Roncal WG, Dyer EL, Gürsoy D, Kording K, and Kasthuri N. From sample to knowledge: towards an integrated approach for neuroscience discovery. *arXiv*, 2016.

[C1] Bergamaschi A, Medjoubi K, Messaoudi C, Marco S, and Somogyi A. Mmx-i: data-processing software for multimodal x-ray imaging and tomography. *Journal od Synchrotron Radiation*, 23:783–794, 2016.

[C2] Kak AC and Slaney M. *Principles of computerized tomographic imaging*. Volume 33. SIAM, 1988.

[C3] Dempster AP, Laird NM, and Rubin DB. Maximum likelihood from incomplete data via the em algorithm. *journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[C4] Münch B, Trtik P, Marone F, and Stampanoni M. Stripe and ring artifact removal with combined wavelet–fourier filtering. *Optics Express*, 17(10):8567–8591, 2009.

[C5] Dowd BA, Campbell GH, Marr RB, Nagarkar VV, Tipnis SV, Axe L, and Siddons DP. Developments in synchrotron x-ray computed microtomography at the national synchrotron light source. In *Proc. SPIE*, volume 3772, 224–236. 1999.

[C6] Francesco De Carlo, Doga Gursoy, Daniel Jackson Ching, Kees Joost Batenburg, Wolfgang Ludwig, Lucia Mancini, Federica Marone, Rajmund Mokso, Daniel M. Pelt, Jan Sijbers, and Mark Rivers. Tomobank: a tomographic data repository for computational x-ray science. *Measurement Science and Technology*, 2017. URL: https://doi.org/10.1088/1361-6501/aa9c19.

[C7] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40(1):120–145, 2011.

[C8] Gursoy D, Hong YP, He K, Hujsak K, Yoo S, Chen S, Li Y, Ge M, Miller LM, Chu YS, De Andrade V, He K, Cossairt O, Katsaggelos AK, and Jacobsen C. Rapid alignment of nanotomography data using joint iterative reconstruction and reprojection. *Scientific Reports*, 2017.

[C9] Paganin D, Mayo SC, Gureyev TE, Miller PR, and Wilkins SW. Simultaneous phase and amplitude extraction from a single defocused image of a homogeneous object. *Journal of Microscopy*, 206(1):33–40, 2002.

[C10] Tilman D, Felix B, and Andreas S. Automated determination of the center of rotation in tomography data. *Journal of the Optical Society of America A*, 23(5):1048–1057, 2006.

[C11] Miqueles EX, Rinkel J, O'Dowd F, and Bermúdez JSV. Generalized titarenko's algorithm for ring artefacts reduction. *Journal of Synchrotron Radiation*, 21(6):1333–1346, 2014.

[C12] Brun F, Pacile S, Accardo A, Kourousias G, Dreossi D, Mancini L, Tromba G, and Pugliese R. Enhanced and flexible software tools for x-ray computed tomography at the italian synchrotron radiation facility elettra. *Fundamenta Informaticae*, 141(2-3):233–243, 2015.

[C13] De Carlo F, Gursoy D, Marone F, Rivers M, Parkinson YD, Khan F, Schwarz N, Vine DJ, Vogt S, Gleber SC, Narayanan S, Newville M, Lanzirotti T, Sun Y, Hong YP, and Jacobsen C. Scientific data exchange: a schema for hdf5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, 2014.

[C14] Manuel GS, Thurman ST, and Fienup JR. Efficient subpixel image registration algorithms. *Optics Letters*, 33(2):156–158, 2008.

[C15] Toby HB, Gürsoy D, De Carlo F, Schwarz N, Sharma H, and Jacobsen CJ. Practices and standards for data and processing at the APS. *Synchrotron Radiation News*, 28(2):15–21, 2015.

[C16] Hudson HM and Larkin RS. Accelerated image reconstruction using ordered subsets of projection data. *Medical Imaging, IEEE Transactions on*, 13(4):601–609, 1994.

[C17] Chang J-H, Anderson JMM, and Votaw JT. Regularized image reconstruction algorithms for positron emission tomography. *Medical Imaging, IEEE Transactions on*, 23(9):1165–1175, 2004.

[C18] Mertens JCE and Chawla JJWN. A method for zinger artifact reduction in high-energy x-ray computed tomography. *Nuclear Instruments and Methods in Physics Research Section A*, 800:82–92, 2015.

[C19] Vogelgesang M, Rota L, Ardila Perez Luis E, Caselle M, Chilingaryan S, and Kopmann A. High-throughput data acquisition and processing for real-time x-ray imaging. In *Proc. SPIE*, volume 9967, 996715–996715–9. 2016. doi:10.1117/12.2237611.

[C20] De Jonge MD, Ryan CG, and Jacobsen C. X-ray nanoprobes and diffraction-limited storage rings: opportunities and challenges of fluorescence tomography of biological specimens. *Journal of Synchrotron Radiation*, 21(5):1031–1047, 2014.

[C21] Rivers ML. Tomorecon: high-speed tomography reconstruction on workstations using multi-threading. In *Proc. SPIE*, volume 8506, 85060U–85060U–13. 2012.

[C22] Vo N, Drakopoulos M, Atwood RC, and Reinhard C. Reliable method for calculating the center of rotation in parallel-beam tomography. *Optics Express*, 22(16):19078–19086, 2014.

[C23] Biçer T, Gürsoy D, Kettimuthu R, De Carlo F, Agrawal G, and Foster IT. Rapid tomographic image reconstruction via large-scale parallelization. In *Lecture Notes in Computer Science*, volume 9233, 289–302. 2015.

[C24] Bhimji W, Bard D, Roumanus M, Paul D, Ovsyannikov A, Friesen B, Bryson M, Correa, Lockwood GK, Tsulaia V, Byna S, Farrell S, Gürsoy D, Daley C, Beckner V, Van Straalen B, Wright NJ, Antypas K, and Prabhat M. Accelerating science with the nersc burts buffer early user program. In *Cray User Group Conference*. 2016.

[C25] Xu W and Feng D. Studying performance of a penalized maximum likelihood method for pet reconstruction on nvidia gpu and intel xeon phi coprocessor. *Proc. 4th Intl Conf. Image Formation in X-ray Computed Tomography*, pages 191–194, 2016.