CHESS Tomography Extension Procedure 1.0.1

Jakob Kaminsky
Cornell University

July 31, 2018

Overview

This module is an extension for tomopy 1.0.1. It allows users to process raw tomographic data through automation and GUIs.

It is designed to decrease the amount of variables necessary to complete reconstructions and increase ease of use, while still maintaining the flexability of tomopy.

By reading this manual and using the template, a new user should be able to complete a full reconstruction with no edits to any code besides the initial inputs.

Dependencies

Software	Version
python	2.7.14
tomopy	1.0.1
matplotlib	2.1.0
ipython	5.4.1
h5py	2.7.0

Initial Data Format

There are two pieces of data that must be loaded in order for proper reconstruction: the tomographic image stack, and theta values for the stack. These should be in the format [elements, rows, angles, columns] and [radians] respectively.

This data should be packaged into an hdf5 file as seperate data sets. This can be done using the save() method included in this module.

Procedure

1 Inputs

In order for the script to run, four parameters must be specified:

1. filename: the filename of the hdf5 data
   ex: '/nfs/chess/aux/user/jk989/data.hdf5'
2. sinogramName: name of tomographic dataset (chosen when using save() function)
   ex: 'tomoImgs'
3. thetaName: name of theta dataset (chosen when using save() function)
   ex: 'theta'
4. moduleFolder: location of tomoFunctions module
   ex: '/nfs/chess/aux/user/jk989'

2 Loading Data

After the inputs are selected, now the module can be imported. After importation, loadHDF5() can be called.

def loadHDF5(fileName,
sinogramName = 'tomoImgs',
thetaName = 'theta')

Loads tomographic data and theta from an hdf5 file. Tomo data should be in the format [elements, rows, angles, cols]. Theta should be an array of radians.

parameter	**type**	**description**
filename	string	file to parse
(optional) sinogramName	string	name of tomographic dataset
(optional) thetaName	string	name of theta dataset

Returns: array ([tomographic data, theta])

3 Determining Bounds

The next step is to launchBoundHelper(). This GUI allows a user to pick the bounds of interest. The radiograph should specify the x element and y layer. It is recommended that the most expansive element and closest to center layer be chosen.

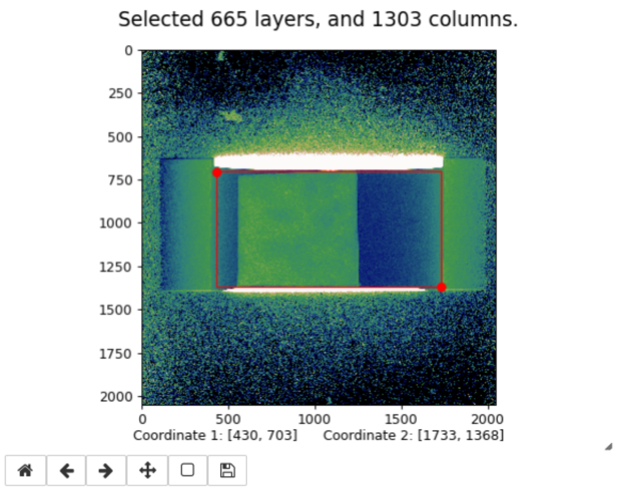
def launchBoundHelper(radiograph,
vMin = None,
vMax = None,
cmap = None,
interpolation = 'none')

Launches GUI which allows users to pick section of sample for evaluation.

parameter	**type**	**description**
radiograph	2D ndarray (tomoImgs[x,:,y,:])	radiograph to display
(optional) vmin/vmax	scalar	lower/upper bound of color spectrum
(optional) cmap	Colormap	(matplotlib.colors.Colormap) function which determines color scheme
(optional) interpolation	string	determines algo for interpolation, see matplotlib.pyplot.imshow for list of algs

Returns: instance of BoundHelper

To chose the sample space, first select the top left point and then bottom right point. If mouse is clicked and dragged, no points will be selected. This is to ensure matplotlib's zoom functionality. Due to the nature of the reconstruction algorithm, leave room on the left and right sides of the sample. See example:



The x coordinates determine imageBounds and the y coordinates determine layerBounds.

4 Determining Reconstruction Values

Now the sample can be reconstructed. The center of the image must be determined in order to get a proper reconstruction. However, because the top and bottom of the sample have different centers, we will need to find the center for the top of layerBounds and the center for the bottom of layerBounds. The center can be determined my changing the center slider in the GUI, trying to optimize the clarity of the image.

Sometimes the extremes of the layerBounds do not contain the sample, so in that case, use the layer input of the GUI to change the layer. However, keep the layer close to the top/bottom edge of the sample to ensure an accurate reconstruction.

```
def launchValHelper(sinograms,
        imageBounds,
```

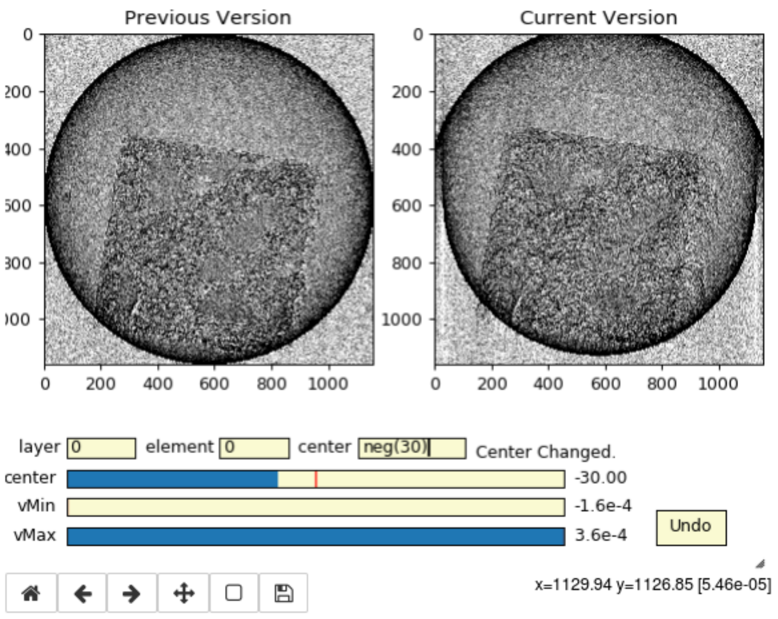
layer,
layerBounds,
theta,
sigma = .1,
ncore = 4,
algorithm = ‘gridrec’,
vmin = None,
vmax = None,
cmap = ‘binary’,
interpolation = ‘none’)

Launches GUI which allows users to pick reconstruction variables.

parameter	**type**	**description**
sinograms	4D ndarray	[elements, rows, angles, cols]
imageBounds	len 2 array	boundary of sample to be evaluated
layer	int	inital layer to be displayed
layerBounds	len 2 array	reconstruction layer bounds
theta	ndarray	list of angles in radians
(optional) sigma	float	damping param in Fourier space
(optional) ncore	int	# of cores that will be assigned
(optional) algorithm	{str, function}	reconstruction algorithm see tomopy.recon.algorithm for list
(optional) vmin/vmax	scalar	lower/upper bound of color spectrum
(optional) cmap	Colormap	(matplotlib.colors.Colormap) function which determines color scheme
(optional) interpolation	string	determines algo for interpolation, see matplotlib.pyplot.imshow for list of algs

Returns: instance of ValHelper

The launched GUI should look as follows:



Then use getImageBounds() and getLayerBounds() to store GUI values into variables.

5 Calculating Centers

Now the centers for each layer must be calculated. If the top center and bottom center are the same, this step can be skipped. Else, use calcCenters() to calculated the centers for all layers in layerBounds.

```
def calcCenters(layerBounds,
                topCenter,
                topLayer,
                bottomCenter,
                bottomLayer)
```

Takes the difference between topCenter and bottomCenter, assumes center movement is linear, and calculates a list of centers for layerBounds.

parameter	**type**	**description**
layerBounds	len 2 array	reconstruction layer bounds
topLayer/bottomLayer	int	layers from top and bottom of stack w/ known centers
topCenter/bottomCenter	scalar	center vals for topLayer/bottomLayer

Returns: array containing extrapolated centers for layerBounds

6 Reconstruction

Now that all parameters have been found, use the reconstruct() method to reconstruct every layer in layerBounds.

```
def reconstruct(sinograms,
centers,
imageBounds,
layers,
theta,
sigma = .1,
ncore = 4,
algorithm = ‘gridrec’)
```

Reconstructs object from projection data. Takes in list [elements, rows, angles, cols] or [rows, angles, cols], and returns ndarray representing a 3D reconstruction.

parameter	**type**	**description**
sinograms	ndarray	3D tomographic data
centers	scalar, array	estimated location(s) of rotation axis
imageBounds	len 2 array	boundary of sample to be evaluated
layers	scalar, len 2 array	single layer or bounds of layers
theta	ndarray	list of angles in radians
(optional) sigma	float	damping param in Fourier space
(optional) ncore	int	# of corse that will be assigned
(optional) alogirthm	str, function	determines algo for interpolation, see matplotlib.pyplot.imshow for list of algs

Returns: ndarray representing multi-elemental 3D reconstructions.

Then use getCenter() and getLayer() to store the GUI values into variables.

7 Post-Reconstruction

There are various tools for visualizing reconstructions. The most simple is plotLayer():

```
def plotLayer(image,
vmin = None,
vmax = None,
area = .6,
cmap = ‘binary’,
interpolation = ‘none’)
```

plots a 2D array

parameter	**type**	**description**
image	2D ndarray	image to display
(optional) vmin/vmax	scalar	lower/upper bound of color spectrum
(optional) area	(0,1]	generality of vmin/vmax estimated (0 includes less pixels, 1 includes all pixels including extremes)
(optional) cmap	Colormap	(matplotlib.colors.Colormap) function which determines color scheme
(optional) interpolation	string	determines algo for interpolation, see matplotlib.pyplot.imshow for list of algs

Returns: AxesImage object (plot)

There is also multiSliceGiffer() and multiSliceViewer(), the former cycles through your reconstructed images automatically, and the latter cycling with keyboard presses {j} and {k}.

```
def multiSliceGiffer(volume,
        cmap = None)
```

Display that cycles through slices of a volume.

parameter	**type**	**description**
volume	3D ndarray	volume to display
(optional) cmap	Colormap	(matplotlib.colors.Colormap) function which determines color scheme

Returns: AxesImage object (plot)

Documentation for multiSliceViewer() is omitted because it is the same as multiSliceGiffer except without the cmap parameter.

8 Saving Data

Once reconstructions have been finalized, save the array as and hdf5 file with save():

```
def save(filename,
        names,
        data)
```

Saves files as hdf5 in specified directory.

parameter	**type**	**description**
fileName	String	desired name of file
names	[String,]	list of desired names of data
data	[ndarray,]	list of data

Returns: None

And the process is complete!

Contact

Any questions or suggestions can be sent to jk989@cornell.edu.