

---

# **tomography***tutorialDocumentation*

***Release 1.0***

**Nesrin Salepci, John Truckenbrodt, Robert Eckhardt**

**Mar 13, 2023**



---

## Contents

---

<b>1</b>	<b>main functions</b>	<b>1</b>
<b>2</b>	<b>ancillary functions</b>	<b>3</b>
<b>3</b>	<b>plotting</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



# CHAPTER 1

---

## main functions

---

The core functions of the tomography package.

`tomography_tutorial.functions.calculate_covariance_matrix` (*img\_stack*, *outname*,  
*kernelsize=10*, *over-*  
*write=False*)

compute the covariance matrix. If the target file already exists and `overwrite=False` this function acts as a simple file reader.

### Parameters

- **img\_stack** (*numpy.ndarray*) – the normalized SLC image stack
- **outname** (*str*) – the name of the file to be written
- **kernelsize** (*int*) – the boxcar smoothing dimension
- **overwrite** (*bool*) – overwrite an existing file? Otherwise it is read from file and returned

**Returns** the covariance matrix

**Return type** *numpy.ndarray*

`tomography_tutorial.functions.capon_beam_forming_inversion` (*covmatrix*, *kz\_array*,  
*outname*, *height=70*,  
*overwrite=False*)

perform the capon beam forming inversion to create the final tomographic result. If the target file already exists and `overwrite=False` this function acts as a simple file reader.

### Parameters

- **covmatrix** (*numpy.ndarray*) – the covariance matrix
- **kz\_array** (*numpy.ndarray*) – the wave number stack
- **outname** (*str*) – the name of the file to be written
- **height** (*int*) – the maximum inversion height
- **overwrite** (*bool*) – overwrite an existing file? Otherwise it is read from file and returned

**Returns** the tomographic array

**Return type** `numpy.ndarray`

`tomography_tutorial.functions.read_data(input, outname, overwrite=False)`  
read the raw input data into a numpy array and write the results

**Parameters**

- **input** (`str` or `list`) – a single image file name or a list of multiple files
- **outname** (`str`) – the name of the file to be written.
- **overwrite** (`bool`) – overwrite an existing file? Otherwise it is read from file and returned

**Returns** an array in 2D (one file) or 3D (multiple files)

**Return type** `numpy.ndarray`

`tomography_tutorial.functions.start(notebook)`

Create a custom copy of the jupyter notebook with a name defined by the user and start it. The notebook is only copied from the package if it does not yet exist. Jupyter notebook files have the extension '.ipynb'. If the defined notebook does not contain this extension it is appended automatically.

**Parameters** **directory** (`str`) – the name of the custom notebook

`tomography_tutorial.functions.topo_phase_removal(img_stack, dem_stack, outname, overwrite=False)`

Removal of Topographical Phase. If the target file already exists and `overwrite=False` this function acts as a simple file reader.

**Parameters**

- **img\_stack** (`numpy.ndarray`) – the SLC image stack
- **dem\_stack** (`numpy.ndarray`) – the image stack containing flat earth and topographic phase
- **outname** (`str`) – the name of the file to be written
- **overwrite** (`bool`) – overwrite an existing file? Otherwise it is read from file and returned

**Returns** the normalized SLC stack

**Return type** `numpy.ndarray`

---

### ancillary functions

---

Additional general functions for the tomography package.

`tomography_tutorial.ancillary.cbfi` (*slice*, *nTrack*, *height*)  
computation of capon beam forming inversion for a single pixel. This function is used internally by the core function `capon_beam_forming_inversion()`.

#### Parameters

- **slice** (*numpy.ndarray*) – an array containing the covariance matrix and wave number for a single pixel
- **nTrack** (*int*) – the number of original SLC files
- **height** (*int*) – the maximum inversion height

**Returns** the tomographic result for one pixel

**Return type** `numpy.ndarray`

`tomography_tutorial.ancillary.geocode` (*data*, *lut\_rg\_name*, *lut\_az\_name*, *outname=None*, *range\_min=0*, *range\_max=None*, *azimuth\_min=0*, *azimuth\_max=None*)

Geocode a radar image using lookup tables. The LUTs are expected to be georeferenced and contain range and azimuth radar coordinates for a specific image data set which is linked to these LUTs. If parameter *data* is a subset of this data set, the pixel coordinates of this subset need to be defined.

#### Parameters

- **data** (*numpy.ndarray*) – the image data in radar coordinates
- **lut\_rg\_name** (*str*) – the name of the range coordinates lookup table file
- **lut\_az\_name** (*str*) – the name of the azimuth coordinates lookup table file
- **outname** (*str* or *None*) – the name of the file to write; if *None*, the geocoded array is returned and no file is written. See function `geowrite()` for details on how the file is written.
- **range\_min** (*int*) – the minimum range coordinate

- **range\_max** (*int*) – the maximum range coordinate
- **azimuth\_min** (*int*) – the minimum azimuth coordinate
- **azimuth\_max** (*int*) – the maximum azimuth coordinate

### Example

```
>>> from osgeo import gdal
>>> from tomography_tutorial.ancillary import geocode
>>> image_name = 'path/to/somedata/image.tif'
>>> lut_rg_name = 'path/to/somedata/lut_rg.tif'
>>> lut_az_name = 'path/to/somedata/lut_az.tif'
>>> outname = 'path/to/somedata/image_sub_geo.tif'
>>> image_ras = gdal.Open(image_name)
>>> image_mat = image_ras.ReadAsArray()
>>> image_ras = None
>>> image_mat_sub = image_mat[0:100, 200:400]
>>> geocode(image_mat_sub, outname, lut_rg_name, lut_az_name, range_min=200,
↳ range_max=400, azimuth_min=0, azimuth_max=100)
```

**tomography\_tutorial.ancillary.geowrite** (*data, outname, reference, indices, nodata=-99*)  
write an array to a file using an already geocoded file as reference. The output format is either GeoTiff (for 2D arrays) or ENVI (for 3D arrays).

#### Parameters

- **data** (*numpy.ndarray*) – the array to write to the file; must be either 2D or 3D
- **outname** (*str*) – the file name
- **reference** (*gdal.Dataset*) – the geocoded reference dataset
- **indices** (*tuple of slices*) – the slices which define the subset of data in reference, i.e. where is the data located within the reference pixel dimensions; see [lut\\_crop\(\)](#)
- **nodata** (*int*) – the nodata value to write to the file

**tomography\_tutorial.ancillary.listfiles** (*path, pattern*)  
list files in a directory whose names match a regular expression

#### Parameters

- **path** (*str*) – the directory to be searched
- **pattern** (*str*) – the regular expression search pattern

**Returns** a list of absolute file names

**Return type** [list](#)

### Example

```
>>> listfiles('/path/to/somedata', 'file[0-9].tif')
['/path/to/somedata/file1.tif', '/path/to/somedata/file2.tif', '/path/to/somedata/
↳ file3.tif']
```

**tomography\_tutorial.ancillary.lut\_crop** (*lut\_rg, lut\_az, range\_min=0, range\_max=None, azimuth\_min=0, azimuth\_max=None*)  
compute indices for subsetting the range and azimuth lookup tables (LUTs). The returned slices describe the minimum LUT subset, which contains all radar coordinates within the range-azimuth subset.



**Parameters**

- **lut\_rg** (*numpy.ndarray*) – the lookup table for range direction
- **lut\_az** (*numpy.ndarray*) – the lookup table for azimuth direction
- **range\_min** (*int*) – first range pixel
- **range\_max** (*int*) – last range pixel
- **azimuth\_min** (*int*) – first azimuth pixel
- **azimuth\_max** (*int*) – last azimuth pixel

**Returns** the pixel indices for subsetting: (ymin:ymax, xmin:xmax)

**Return type** tuple of slices

tomography\_tutorial.ancillary.**normalize** (*slice*)  
normalize a 1D array by its minimum and maximum values:

$$y = \frac{x - \min(x)}{\max(x) - \min(x)}$$

**Parameters** **slice** (*numpy.ndarray*) – the 1d input array to be normalized

**Returns** the normalized array

**Return type** *numpy.ndarray*



Plotting utilities for the Jupyter notebook.

**class** tomography\_tutorial.plotting.**DataViewer** (*slc\_list*, *phase\_list*, *kz\_list*, *slc\_stack*,  
*phase\_stack*, *kz\_stack*)  
 functionality for displaying the input data (SLC, topographic phase and wave number)

### Parameters

- **slc\_list** (*list of str*) – the names of the SLC input files
- **phase\_list** (*list of str*) – the names of the topographic phase input files
- **kz\_list** (*list of str*) – the names of the Kappa-Zeta wave number input files
- **slc\_stack** (*numpy.ndarray*) – the SLC images
- **phase\_stack** (*numpy.ndarray*) – the topographic phase images
- **kz\_stack** (*numpy.ndarray*) – the wave number images

**class** tomography\_tutorial.plotting.**GeoViewer** (*filename*, *cmap='jet'*,  
*band\_indices=None*)  
 plotting utility for displaying a geocoded image stack file.

On moving the slider, the band at the slider position is read from the file and displayed.

### Parameters

- **filename** (*str*) – the name of the file to display
- **cmap** (*str*) – the color map for displaying the image. See `matplotlib.pyplot.imshow()`.
- **band\_indices** (*list*) – a list of indices for renaming the individual bands in *filename* such that one can scroll through the range of inversion heights, e.g. -70:70, instead of the raw band indices, e.g. 1:140. The number of unique elements must be of same length as the number of bands in *filename*.

**class** tomography\_tutorial.plotting.**Tomographypplot** (*capon\_bf\_abs*, *caponnorm*)  
 functionality for creating the main tomography\_tutorial analysis plot

### Parameters

- **capon\_bf\_abs** (*numpy.ndarray*) – the absolute result of the capon beam forming inversion
  - **caponnorm** (*numpy.ndarray*) – the normalized version of *capon\_bf\_abs*; see function *normalize()*.
- **genindex**

### t

`tomography_tutorial.ancillary`, 3  
`tomography_tutorial.functions`, 1  
`tomography_tutorial.plotting`, 7



## C

`calculate_covariance_matrix()` (in module *tomography\_tutorial.functions*), 1

`capon_beam_forming_inversion()` (in module *tomography\_tutorial.functions*), 1

`cbfi()` (in module *tomography\_tutorial.ancillary*), 3

## D

`DataViewer` (class in *tomography\_tutorial.plotting*), 7

## G

`geocode()` (in module *tomography\_tutorial.ancillary*), 3

`GeoViewer` (class in *tomography\_tutorial.plotting*), 7

`geowrite()` (in module *tomography\_tutorial.ancillary*), 4

## L

`listfiles()` (in module *tomography\_tutorial.ancillary*), 4

`lut_crop()` (in module *tomography\_tutorial.ancillary*), 4

## N

`normalize()` (in module *tomography\_tutorial.ancillary*), 5

## R

`read_data()` (in module *tomography\_tutorial.functions*), 2

## S

`start()` (in module *tomography\_tutorial.functions*), 2

## T

`tomography_tutorial.ancillary` (module), 3

`tomography_tutorial.functions` (module), 1

`tomography_tutorial.plotting` (module), 7

`Tomographyplot` (class in *tomography\_tutorial.plotting*), 7

`topo_phase_removal()` (in module *tomography\_tutorial.functions*), 2